Introduction to Artificial Intelligence

丁尧相
浙江大学

Fall & Winter 2022
Week 2

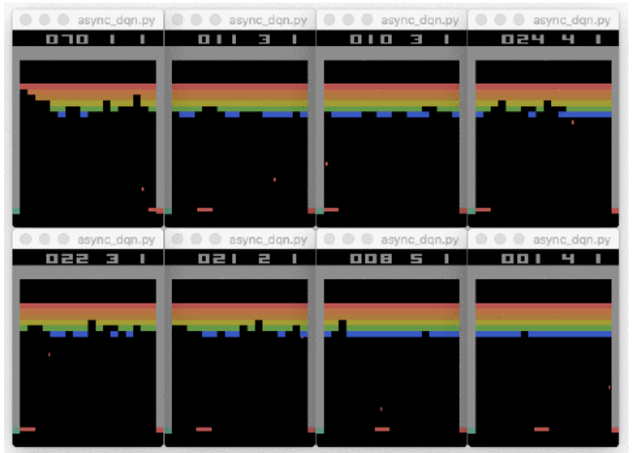
Announcements

- We will have NO lab class this Wednesday.
  - We will release materials of introductory Python.
- The problems for lecture 2 will be released this Wednesday.
- Problem set 1 will due before Lecture 5.

Outline: Decision Making (I)

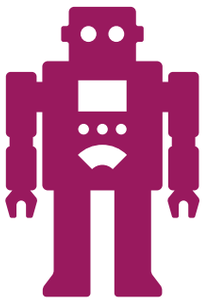
- Uninformed search
  - Breadth-first search
  - Depth-first search
- Informed search
  - Best-first search
  - A* search
- Take-Home Messages

Decision Making



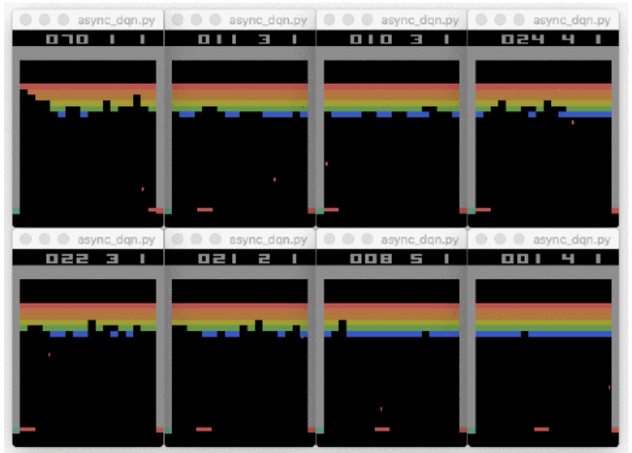
- Conduct action in any state of an environment.

agent



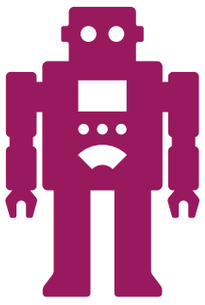
environment

Decision Making



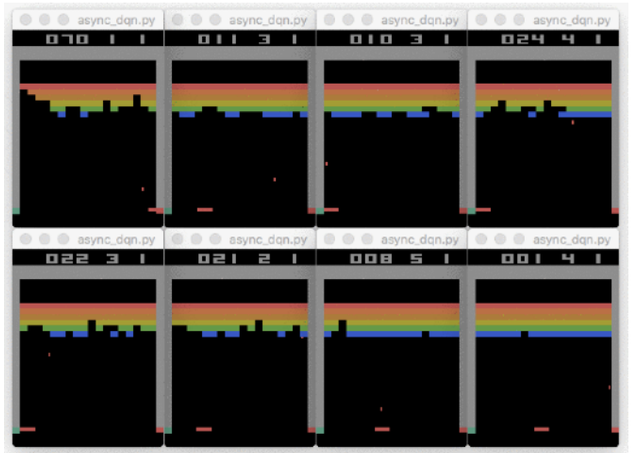
- Conduct action in any state of an environment.

agent

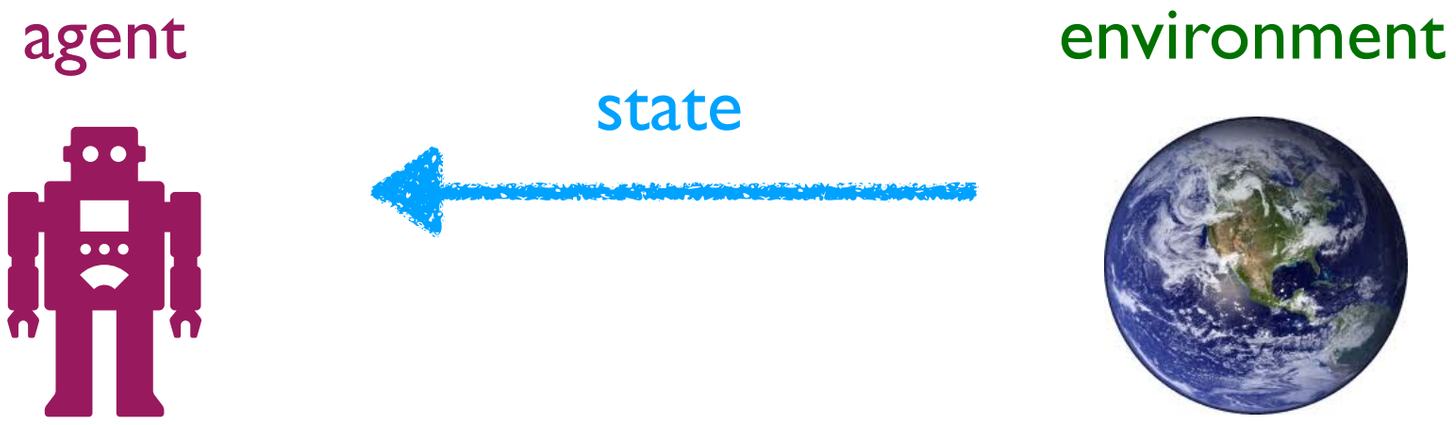


environment

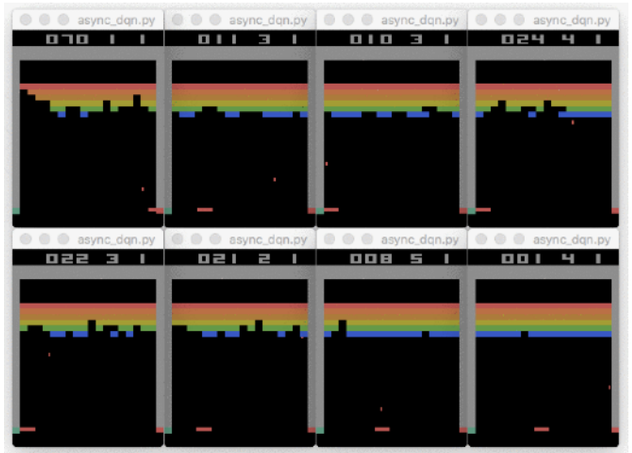
Decision Making



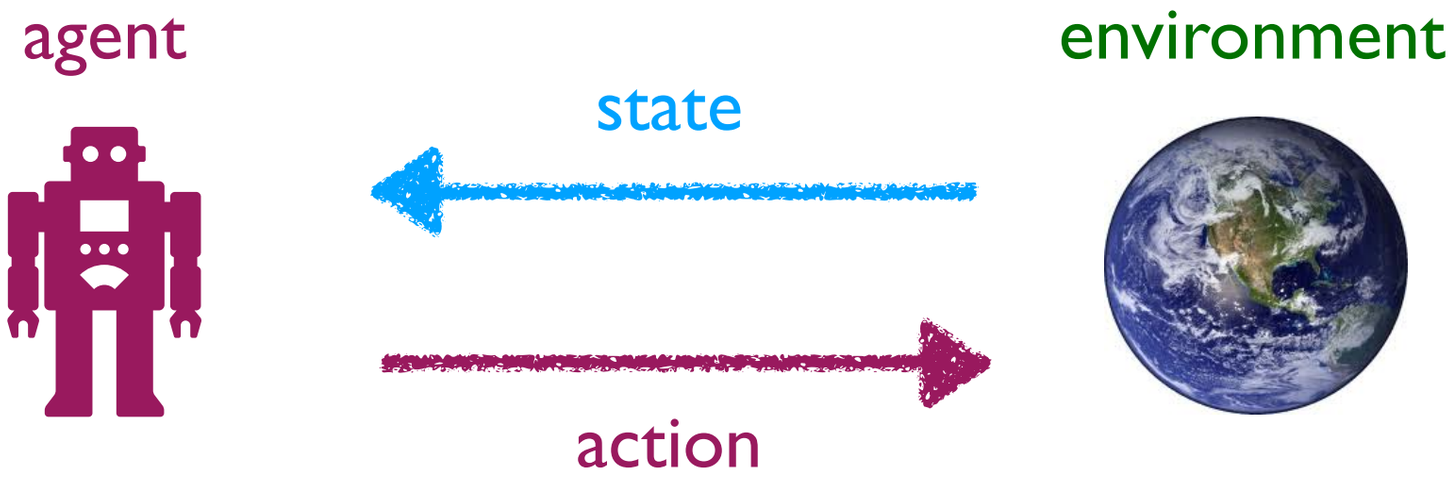
- Conduct action in any state of an environment.



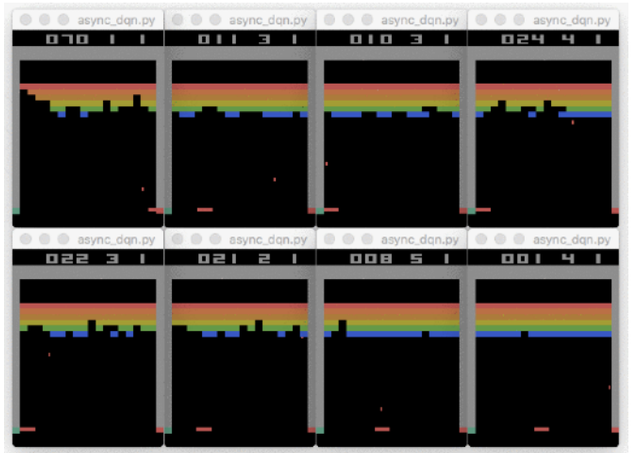
Decision Making

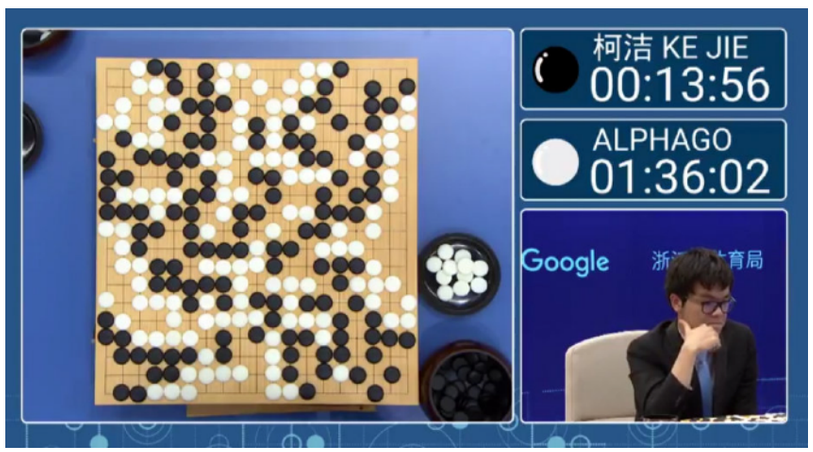


- Conduct action in any state of an environment.

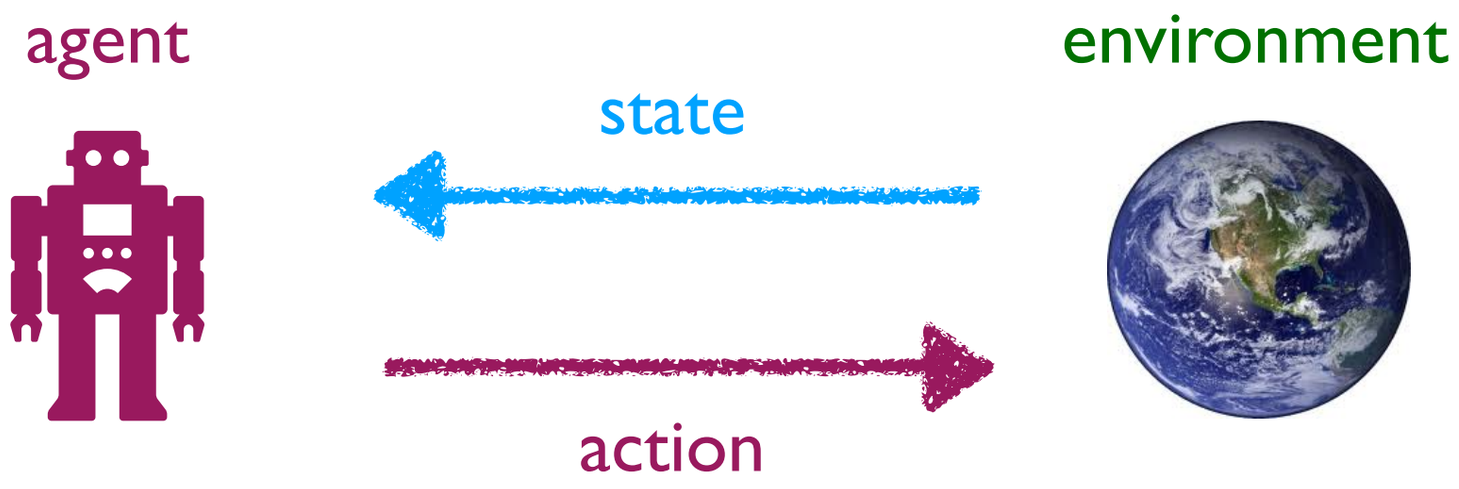


Decision Making





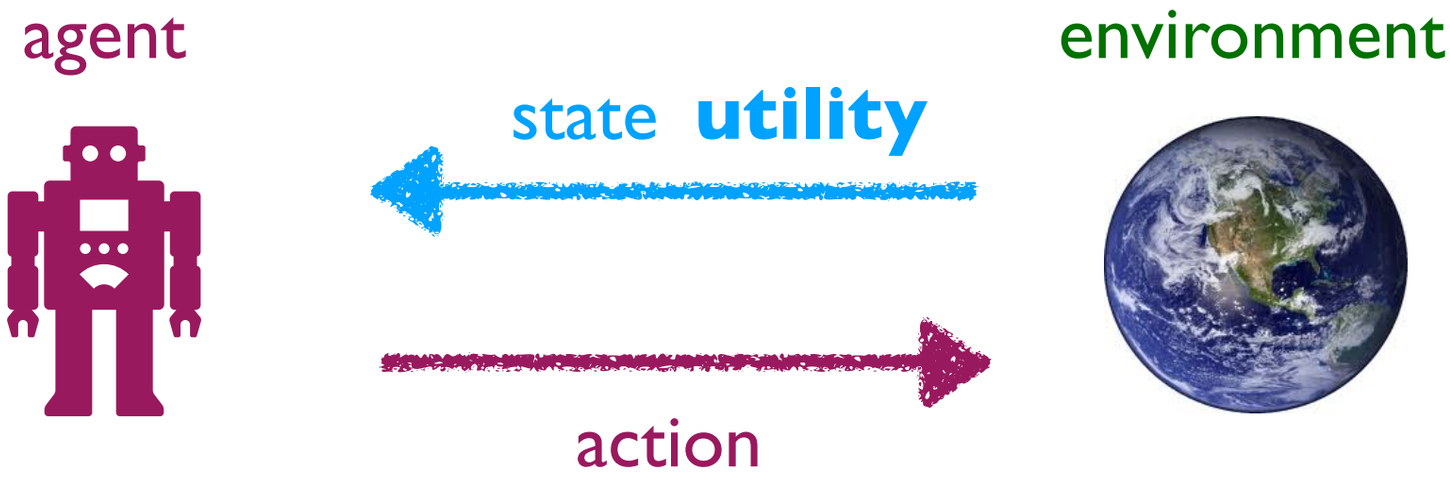
- Conduct action in any state of an environment.



In most problems, the agent needs to do a sequence of actions w.r.t. a sequence of states.

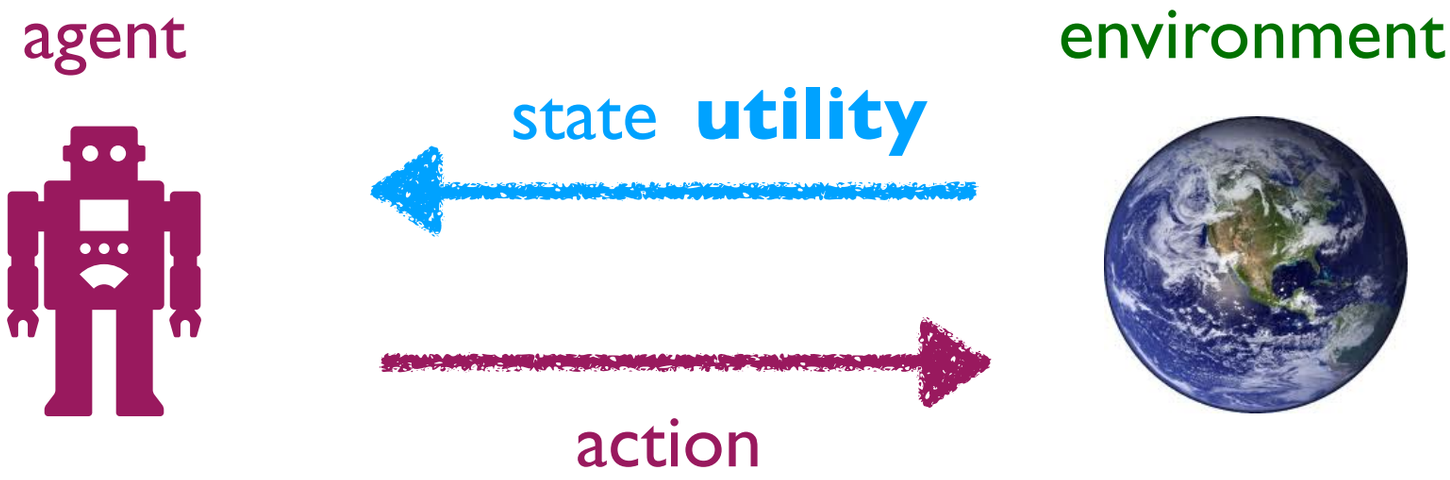
Reflex vs. Problem-Solving Agents

Problem-Solving Agents



- What is the best actions that the agent can take?
  - Reach a goal with the minimal cost.
  - Obtain the most accumulative utilities along the sequence.

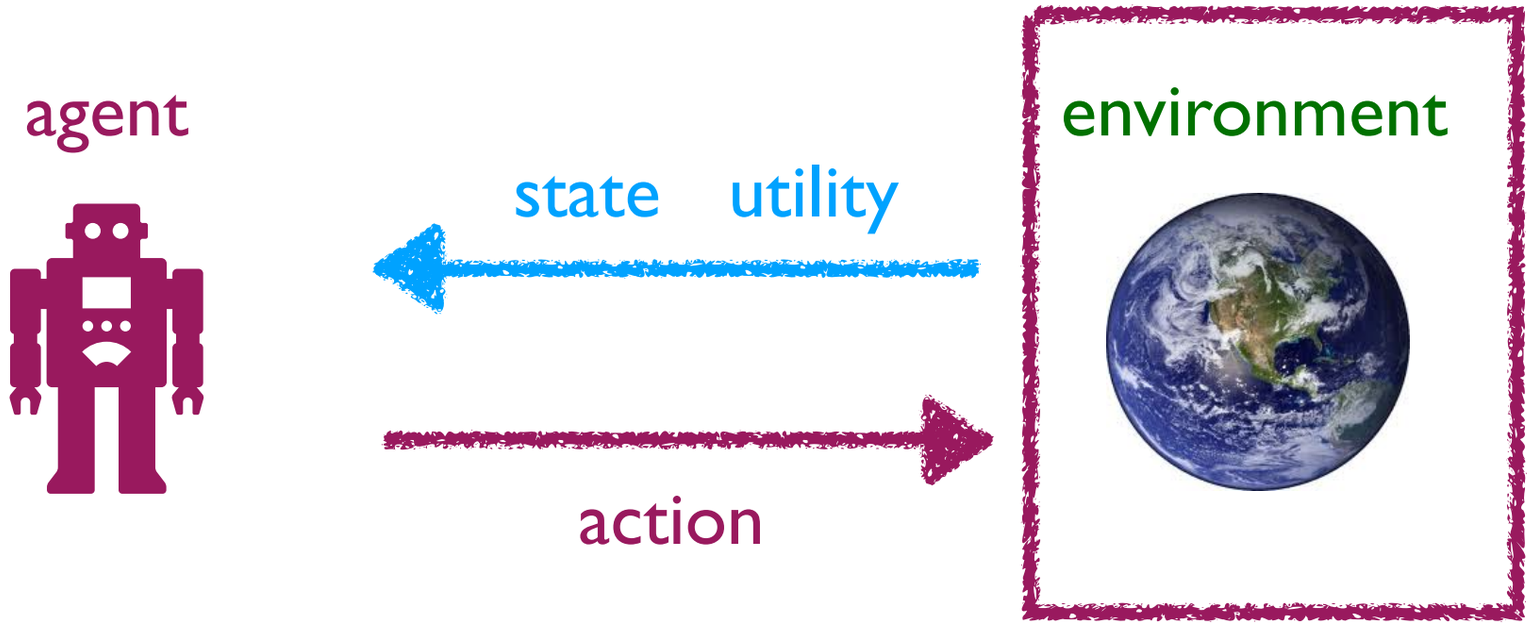
Problem-Solving Agents



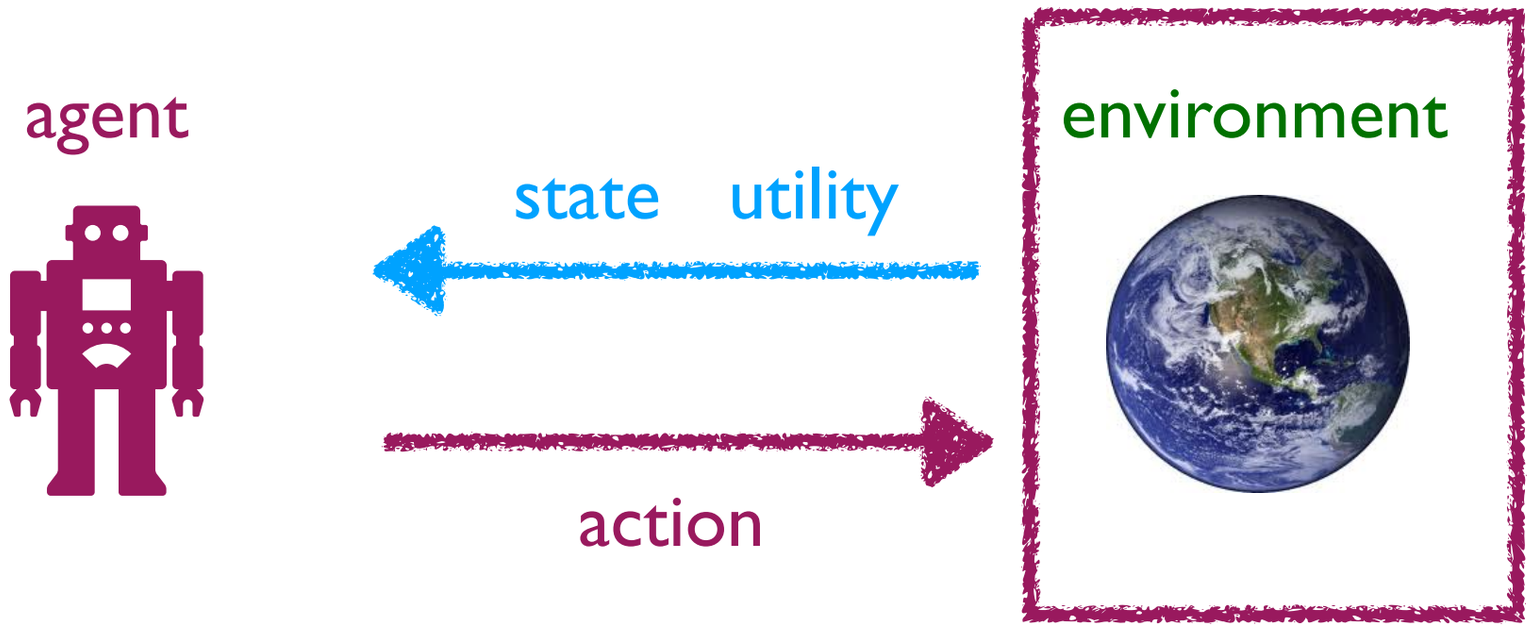
- What is the best actions that the agent can take?
  - Reach a goal with the minimal cost.
  - Obtain the most accumulative utilities along the sequence.

Goal-reaching and utility-maximizing agents are rational.
A problem-solving agent usually needs to care about future utilities instead of only the current one.

Model of the Environment



Model of the Environment



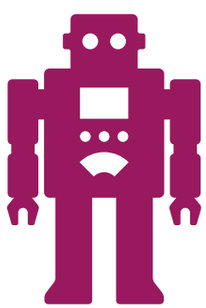
To make decisions in the environment, the agent usually needs a model of the environment to know how the things go on.
Where does this model come from?
Given by the problem (external) or built by the agent? (internal)

Internal vs. External Model

In the decision-making lectures (recent three), we will assume that the agent can directly use the external model. We will consider building the internal model in the knowledge reasoning lectures!

internal model

agent



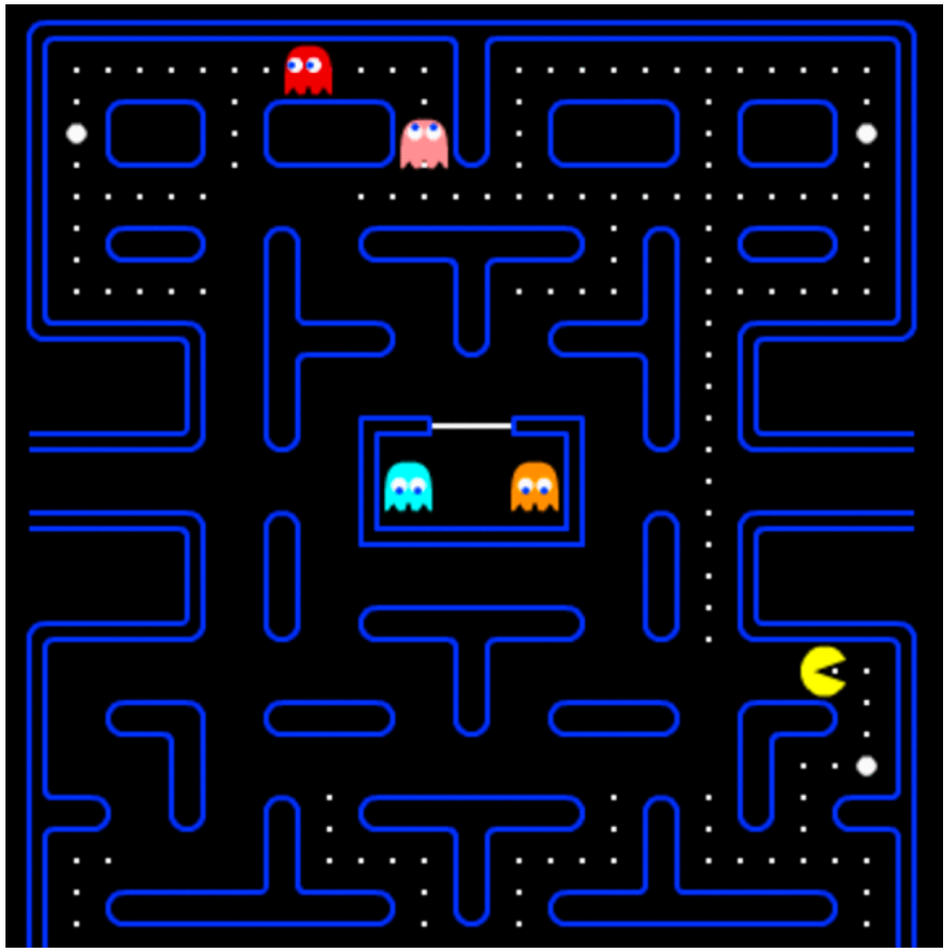
partial state

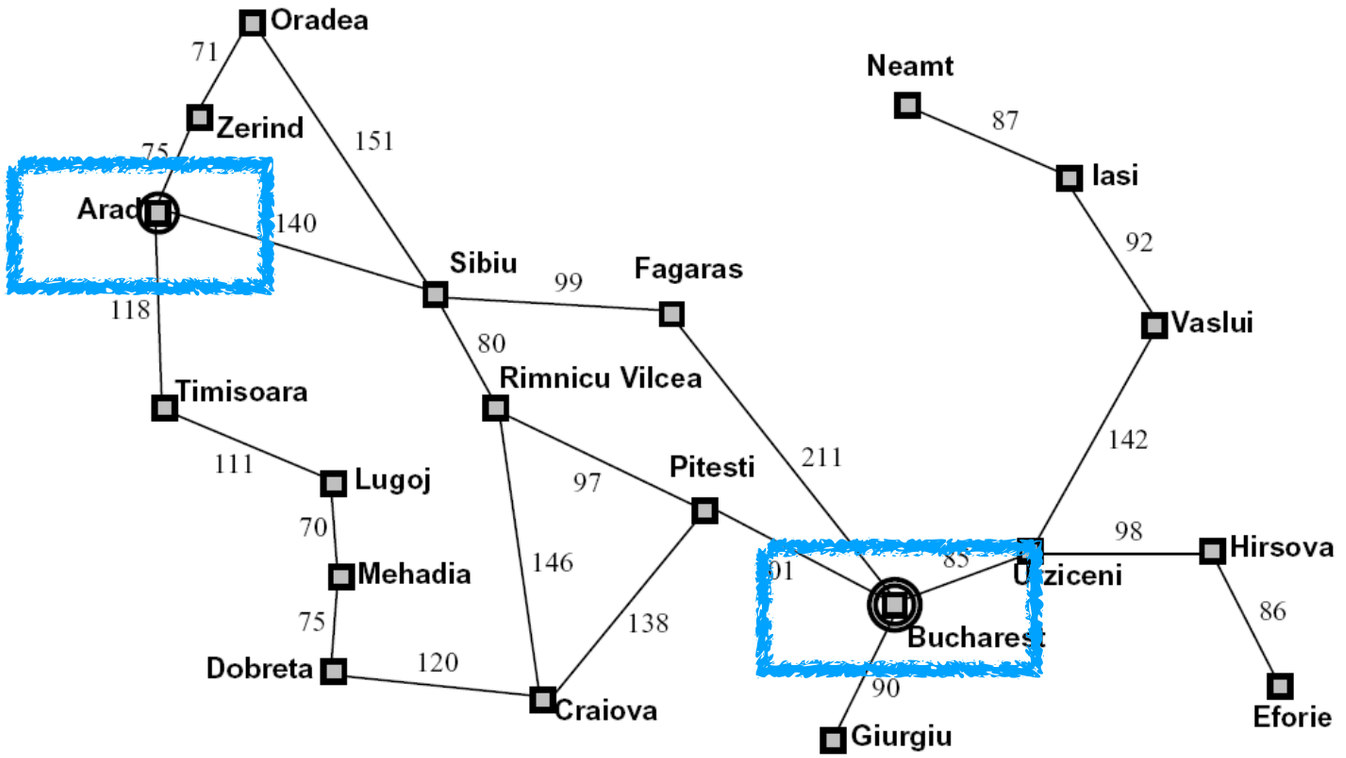
utility

action

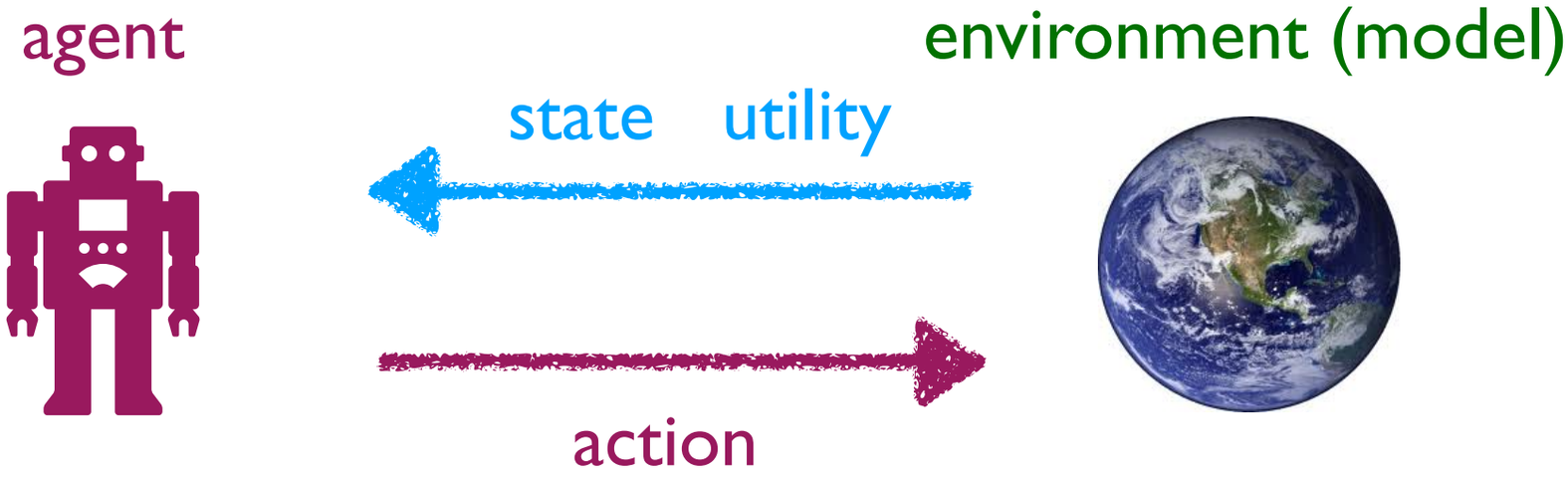
environment (external model)

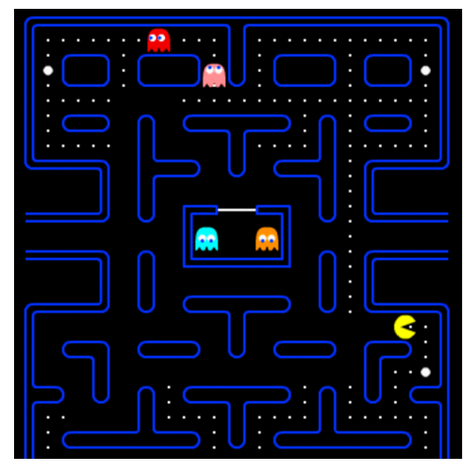
Search Problem



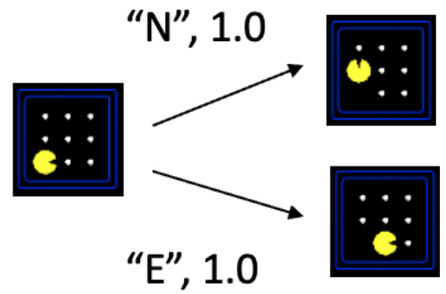


Search Problem

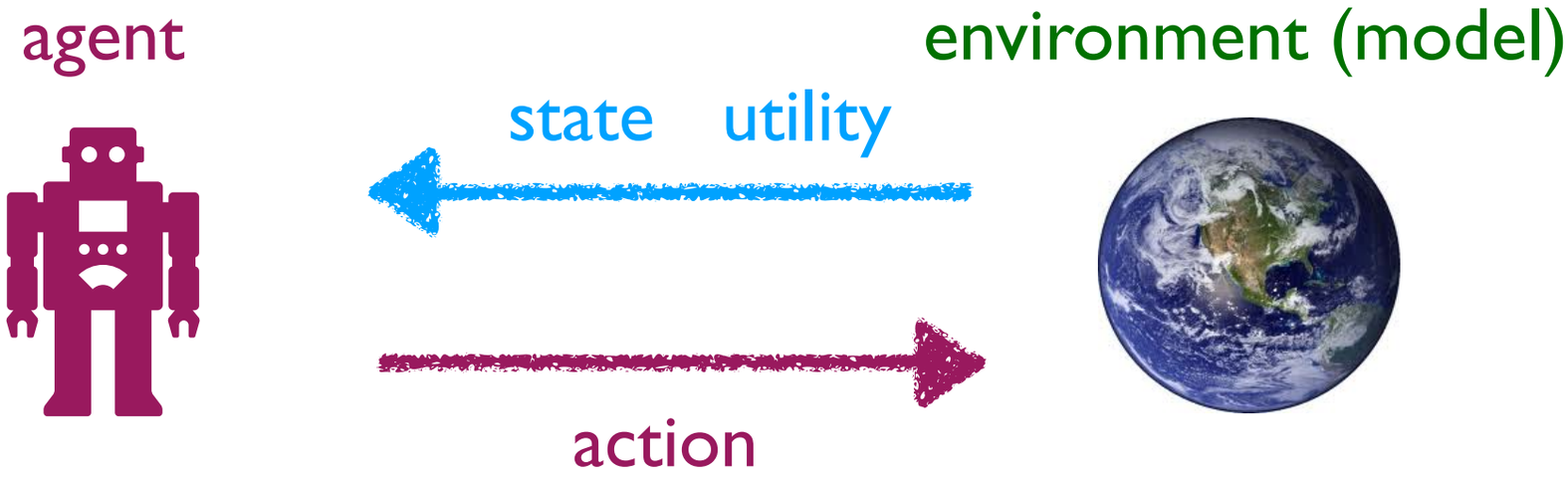


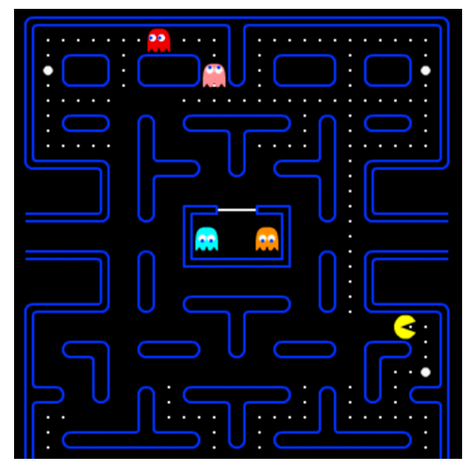


- Problem-solving agents usually solve search problems with the following formulation of model:
  - A state space: 
  - A transition function: state $\times$ action $\rightarrow$ (state, utility)
  - A start state and goal test

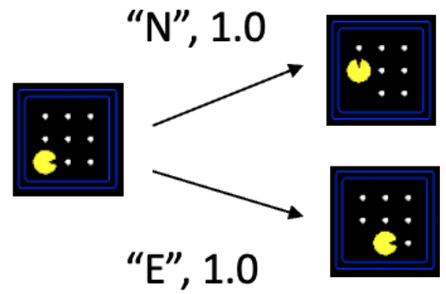


Search Problem



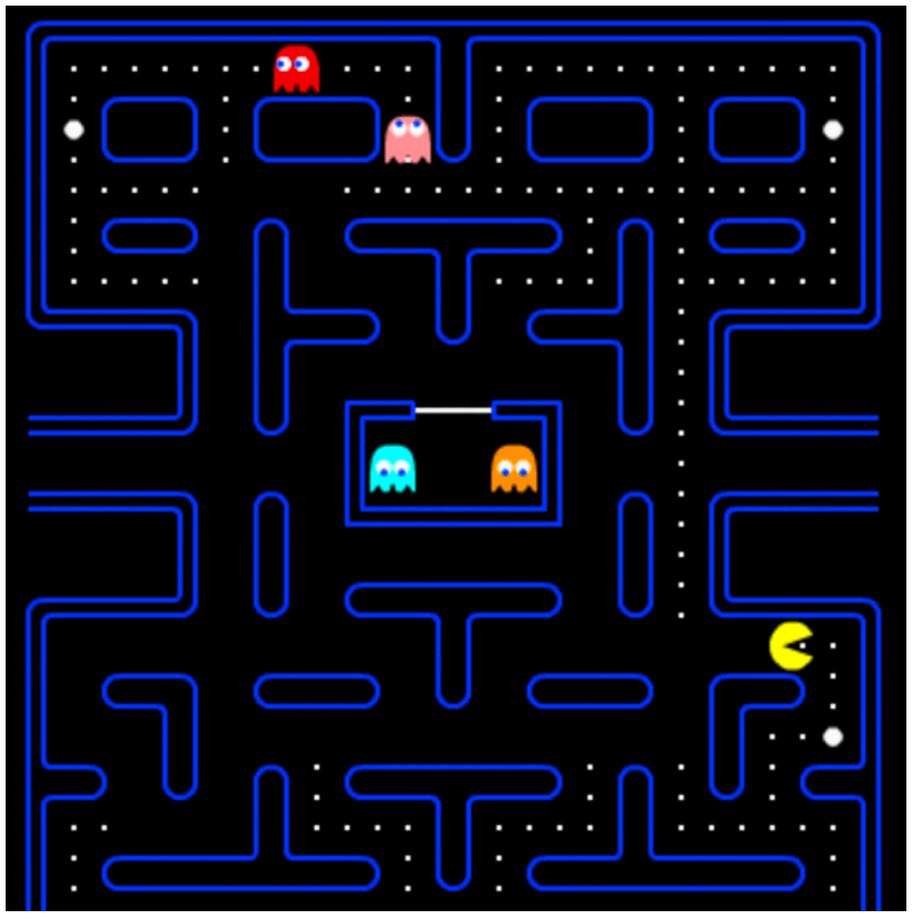


- Problem-solving agents usually solve search problems with the following formulation of model:
  - A state space: 
  - A transition function: state $\times$ action $\rightarrow$ (state, utility)
  - A start state and goal test



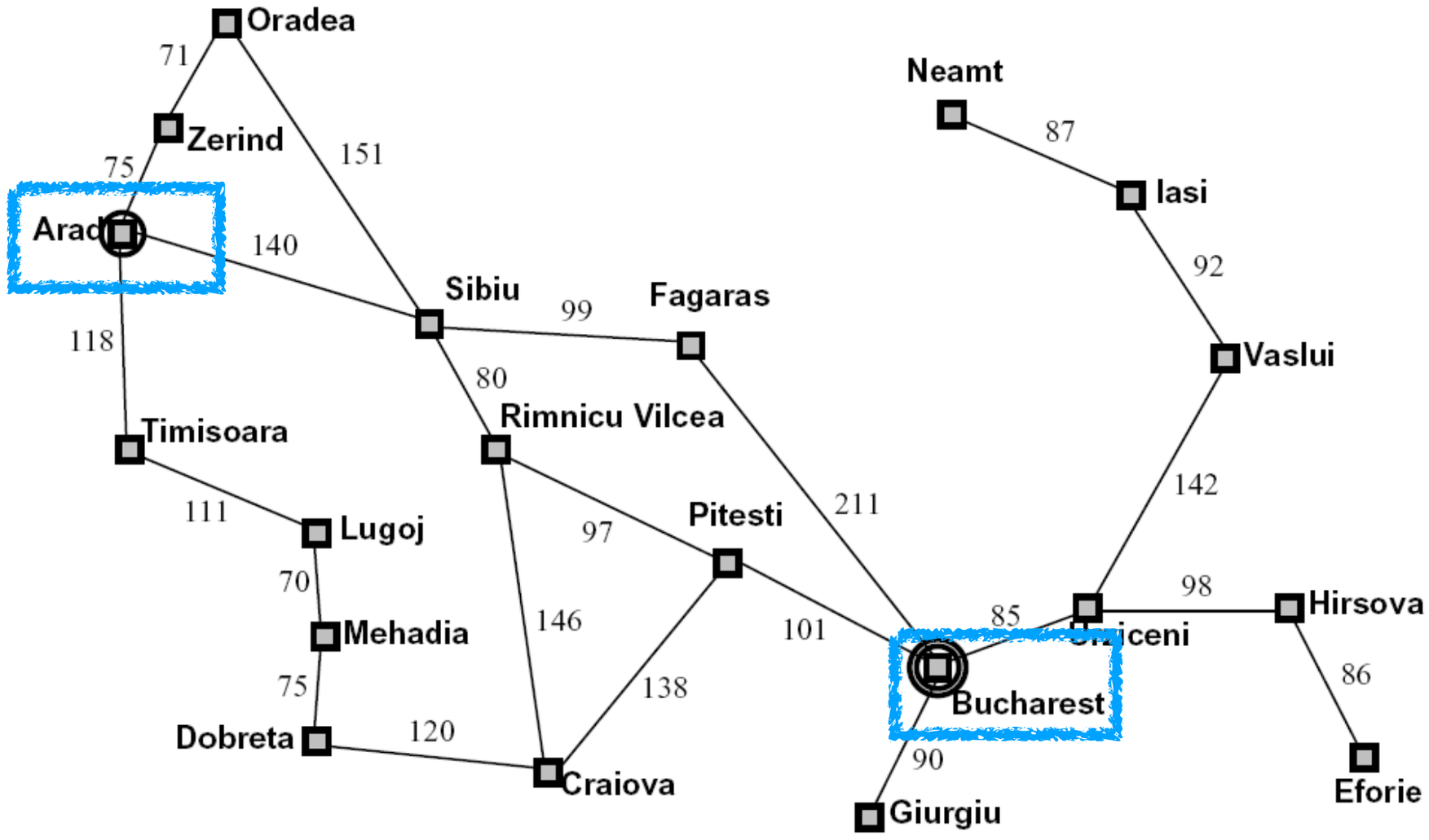
The solution is a sequence of actions that lead the agent goes from the start to the goal state.

Example: Pacman



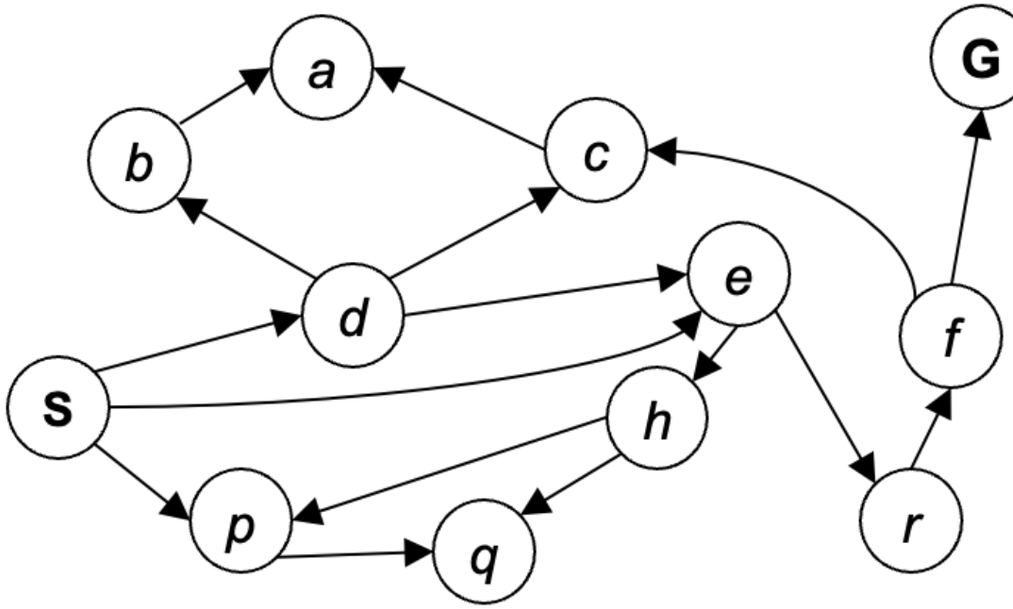
What are the state space, transition function,
start state, and goal test here?

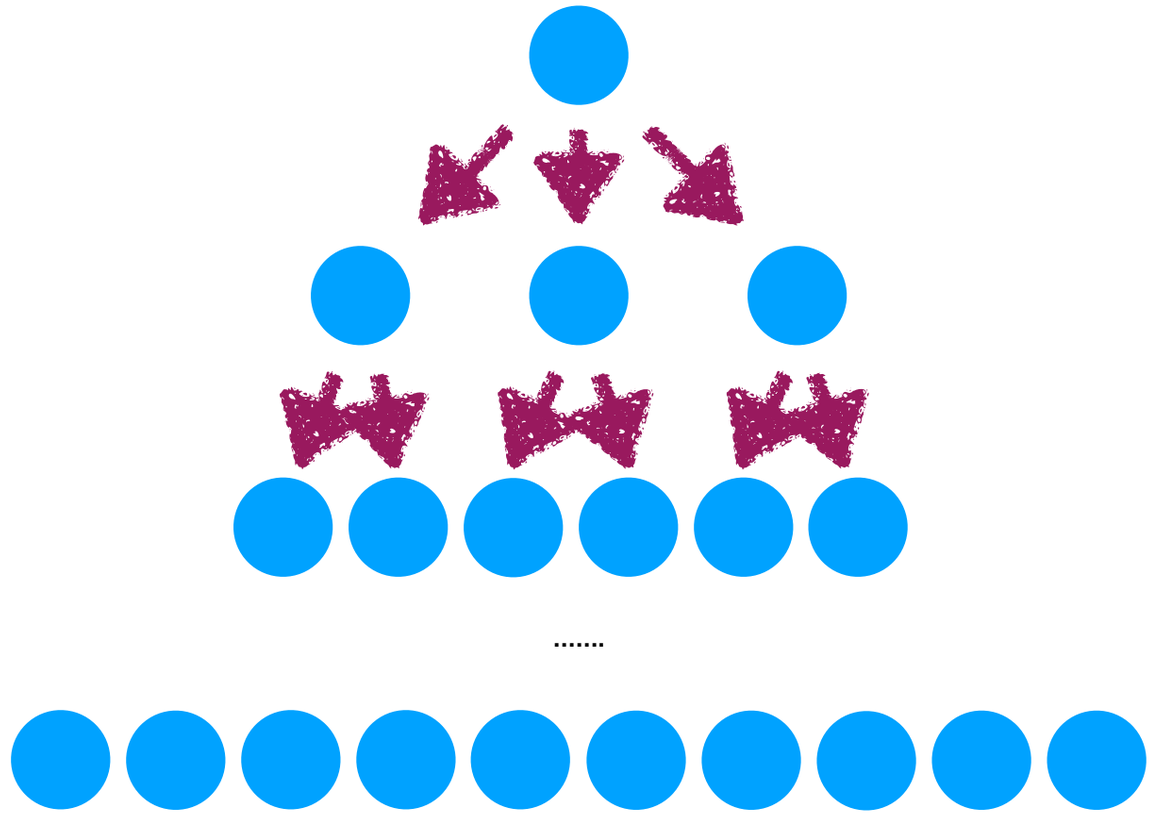
Example: Traveling in Romania



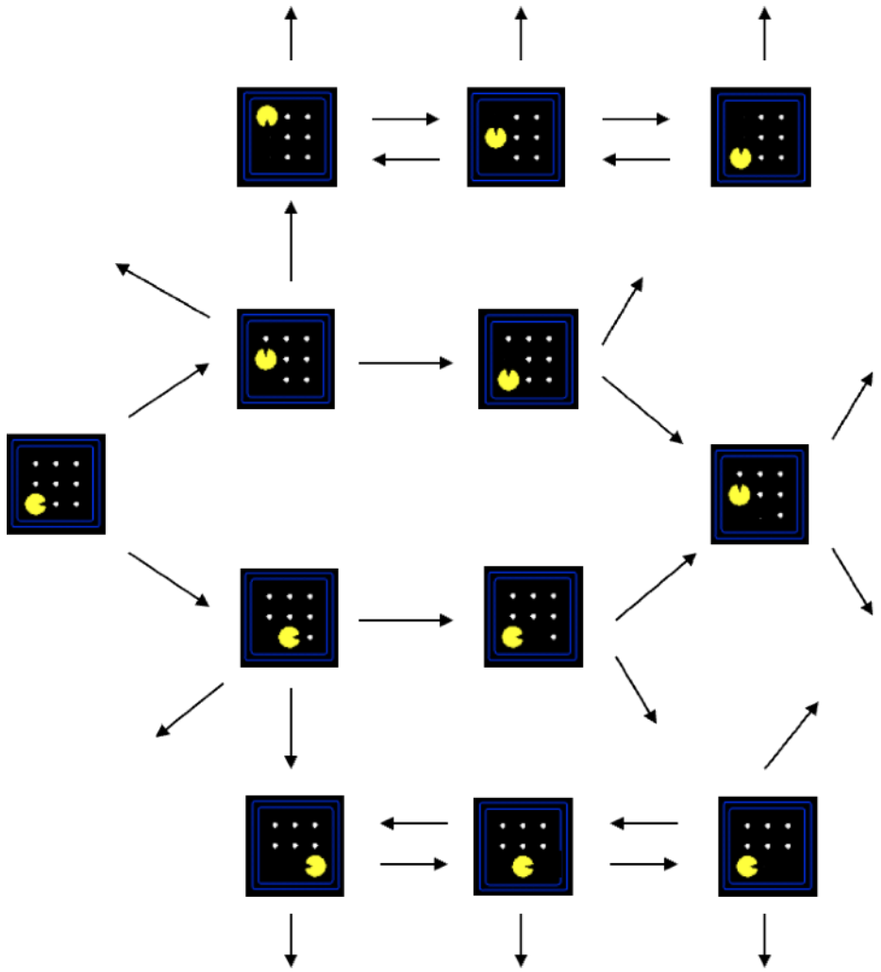
What are the state space, transition function, start state, and goal test here?

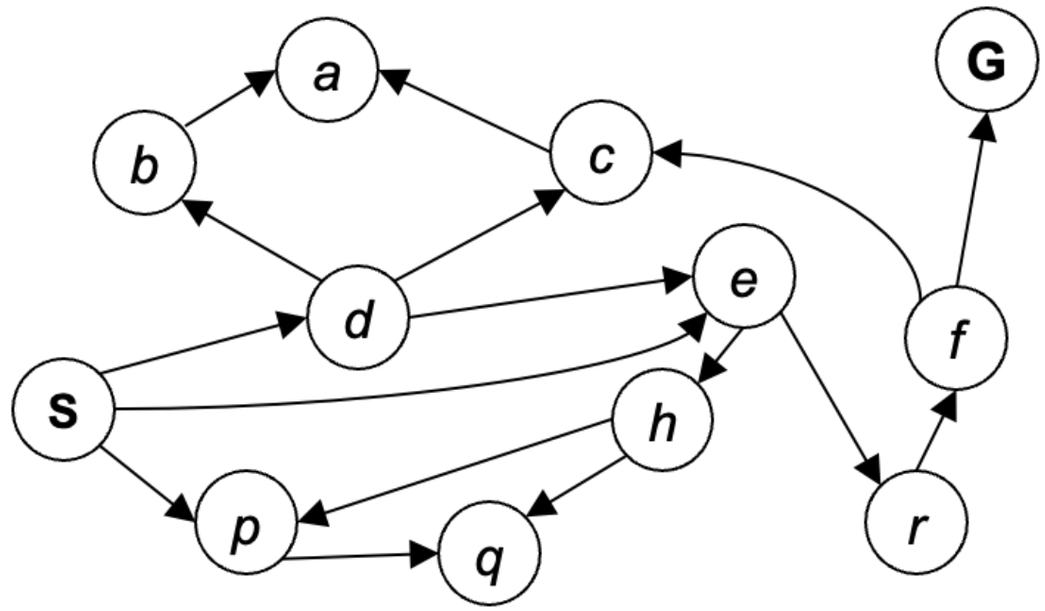
State Space Graph & Search Tree





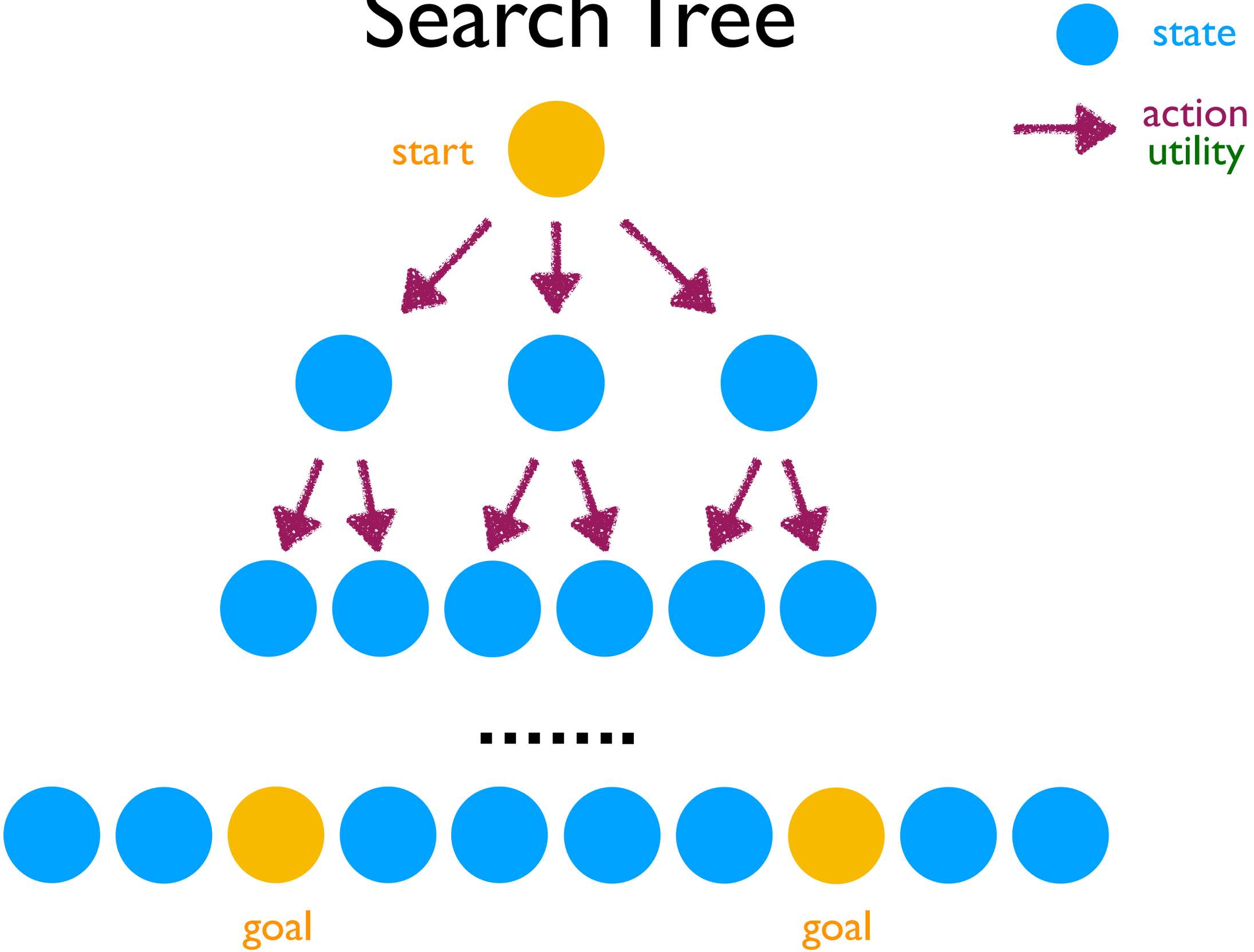
State Space Graph



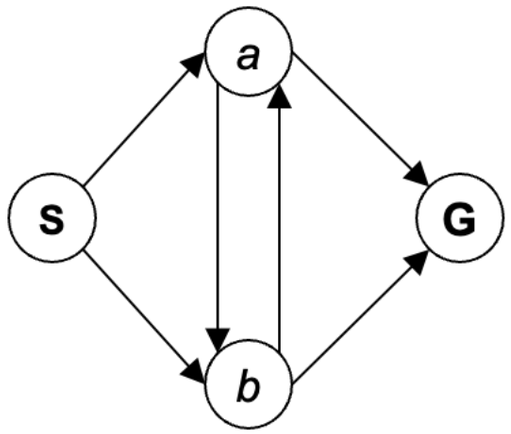


The nodes are states (start and goal states included).
The directed edges are transitions.

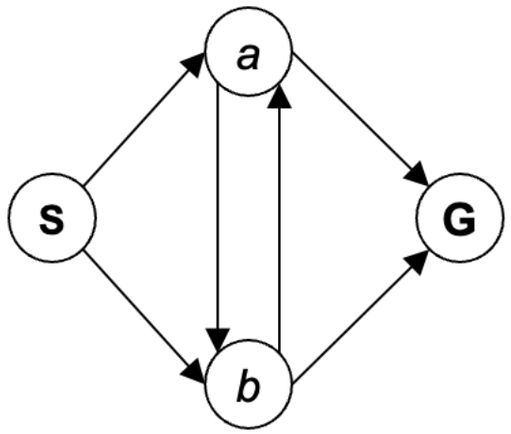
Search Tree



What is the Depth of the Search Tree for the Graph?

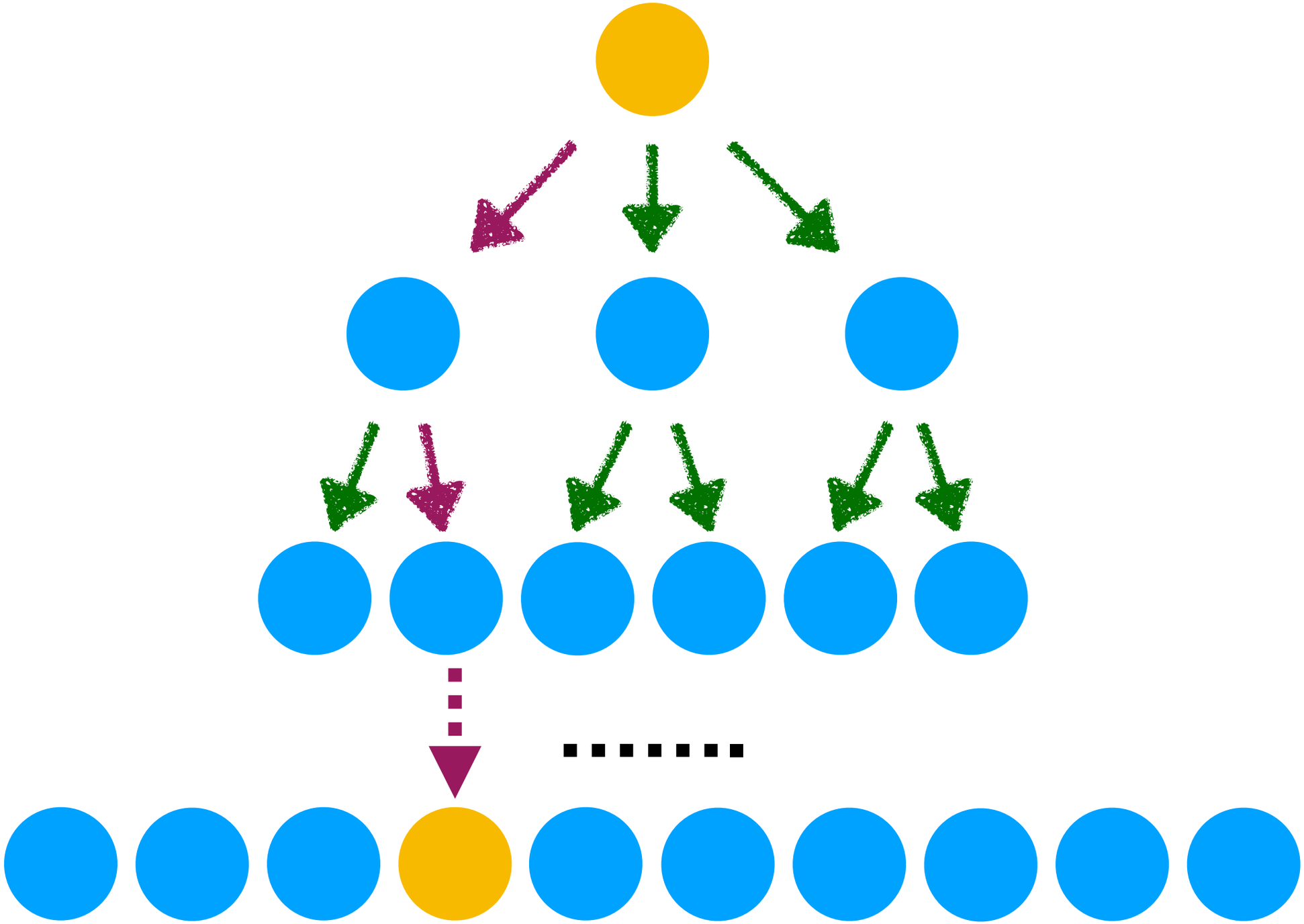


What is the Depth of the Search Tree for the Graph?



There may be a lot of repeated structures in a search tree.

Search Algorithms: the Basic Strategy for Decision Making



Search Algorithms: the Basic Strategy for Decision Making

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
end
```

Search Algorithms: the Basic Strategy for Decision Making

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
end
```

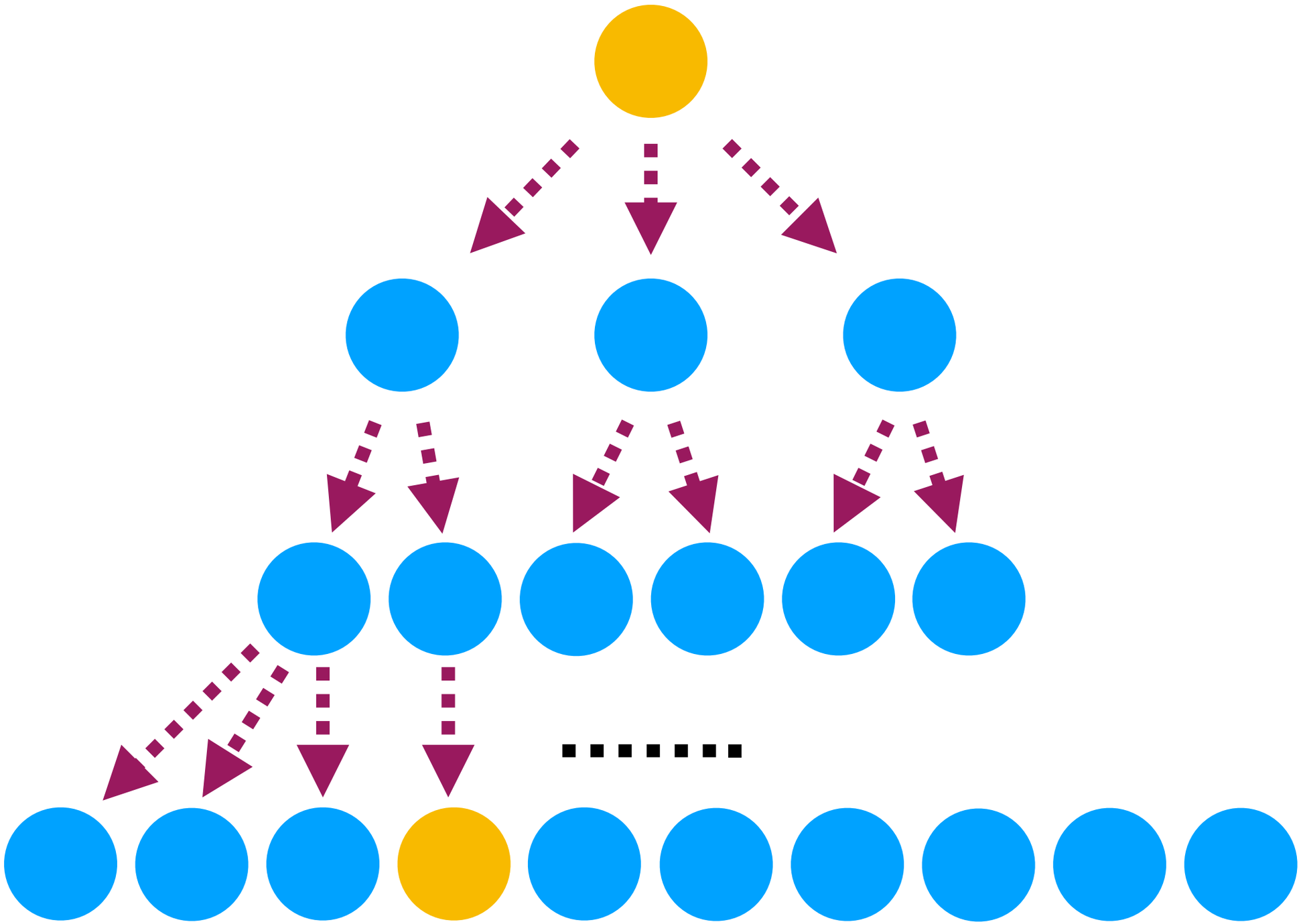
The expansion strategy is essential.

Outline: Decision Making (I)

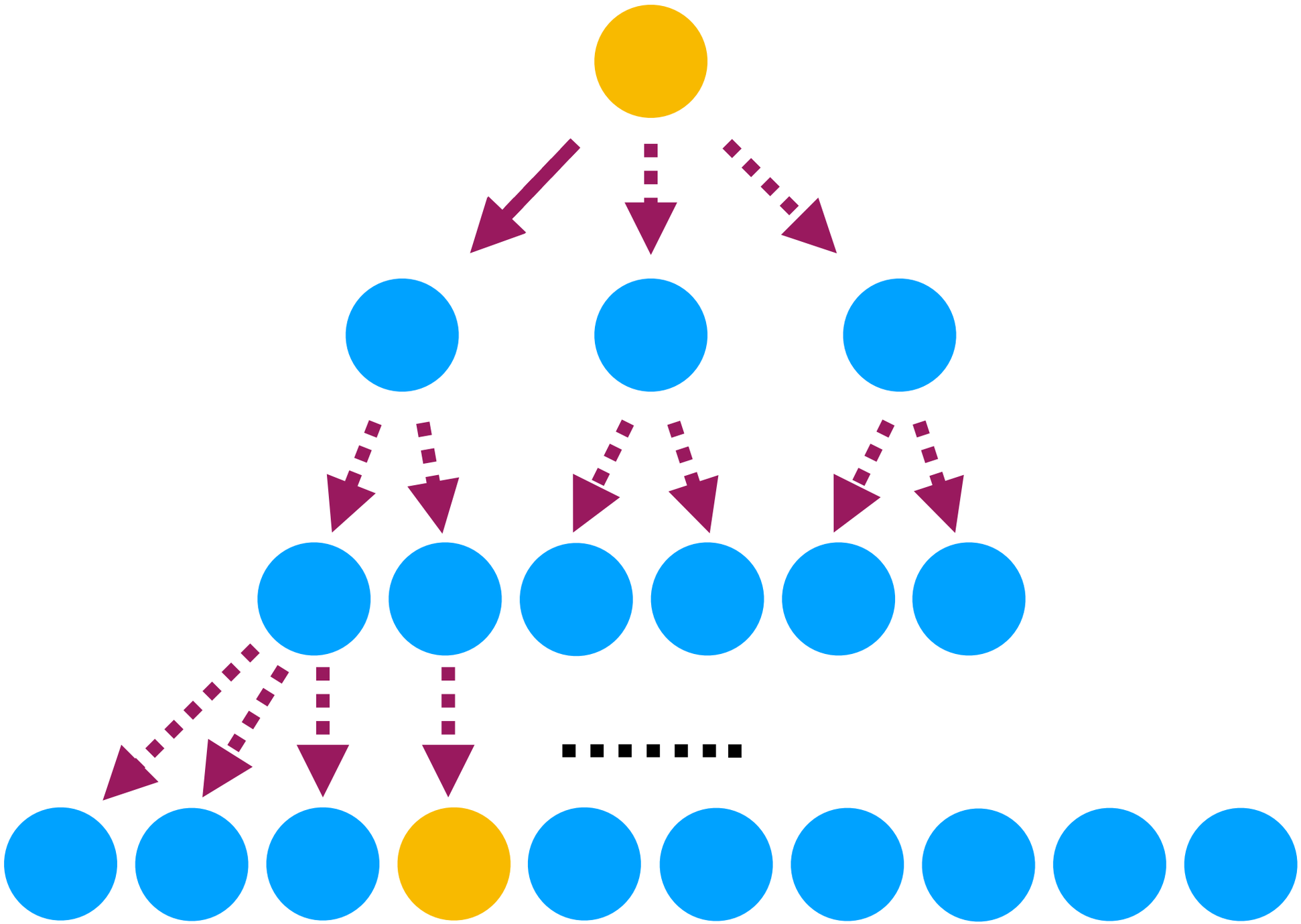
- Uninformed search
  - Breadth-first search
  - Depth-first search
- Informed search
  - Best-first search
  - A* search
- Take-Home Messages

Notice: Assume uniform cost for any action until P. 27.

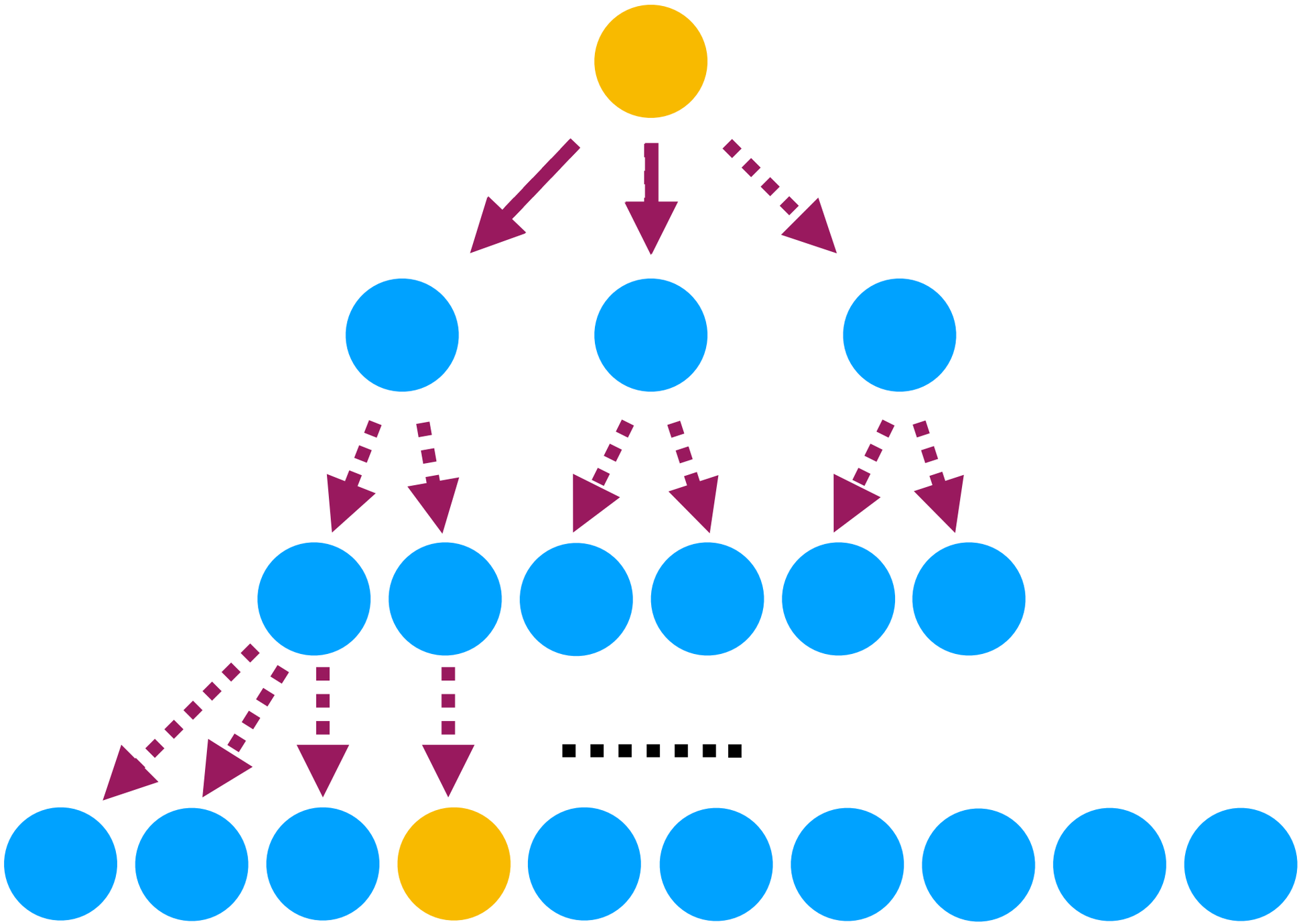
Breath-First Search (BFS)



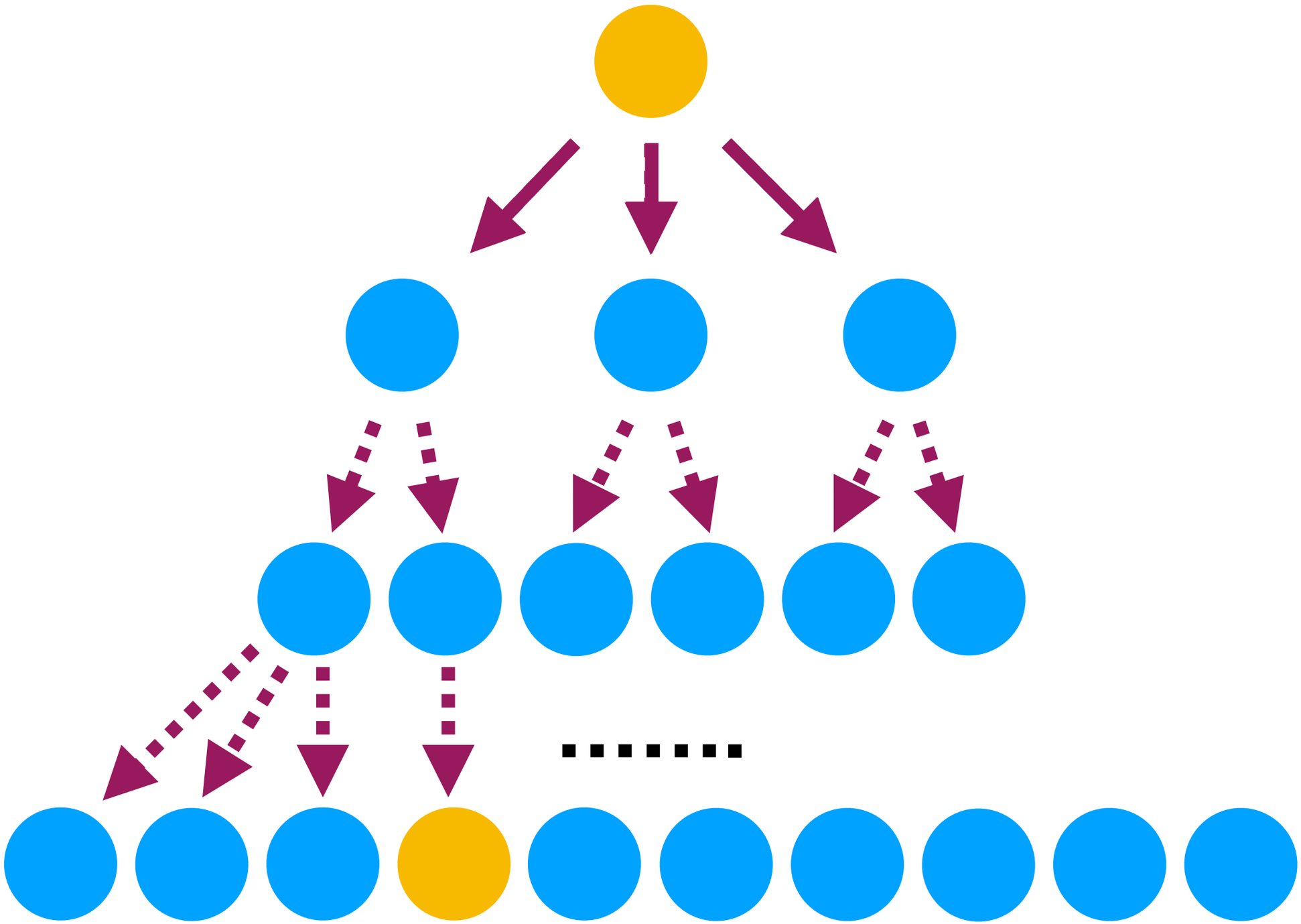
Breath-First Search (BFS)



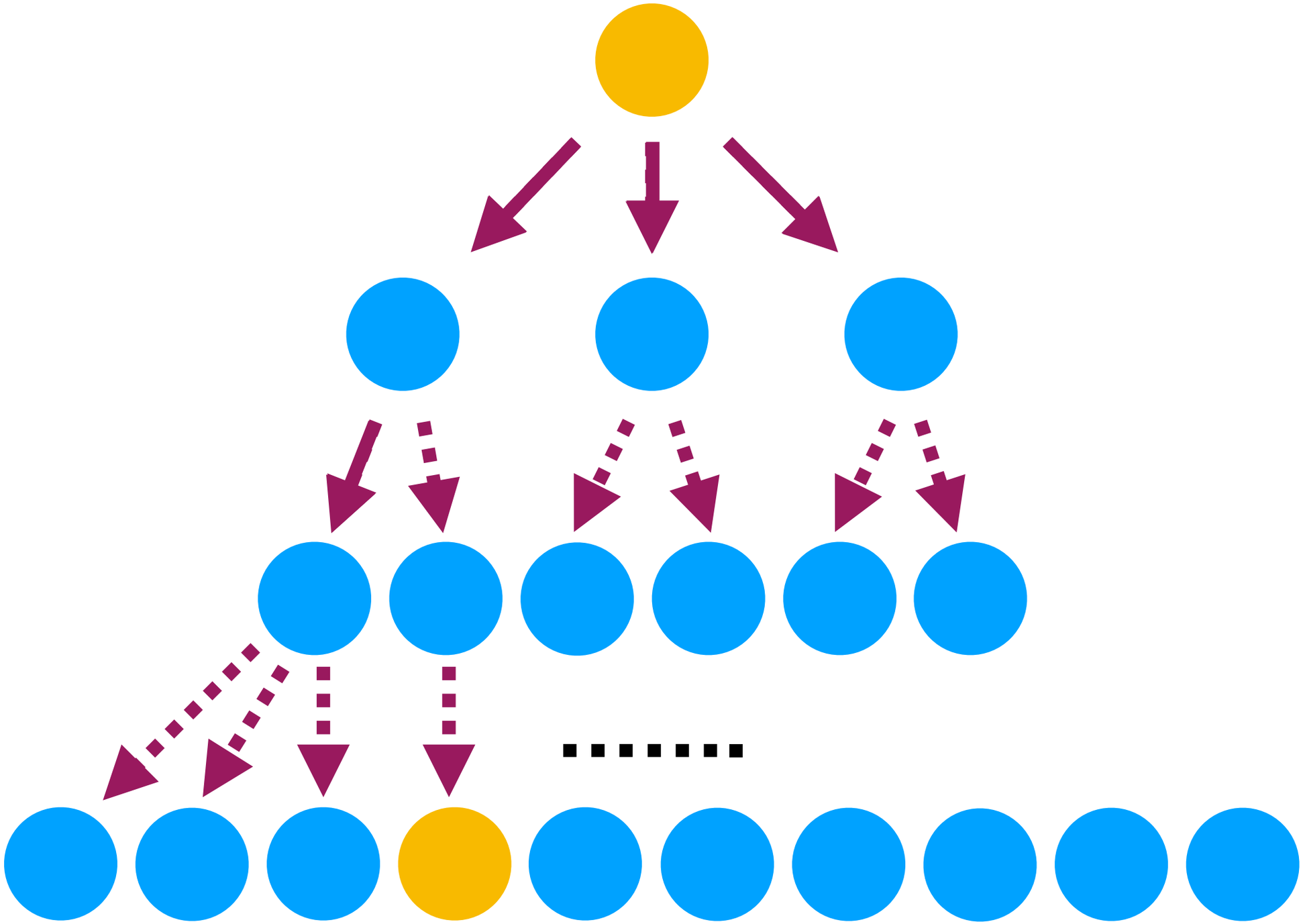
Breath-First Search (BFS)



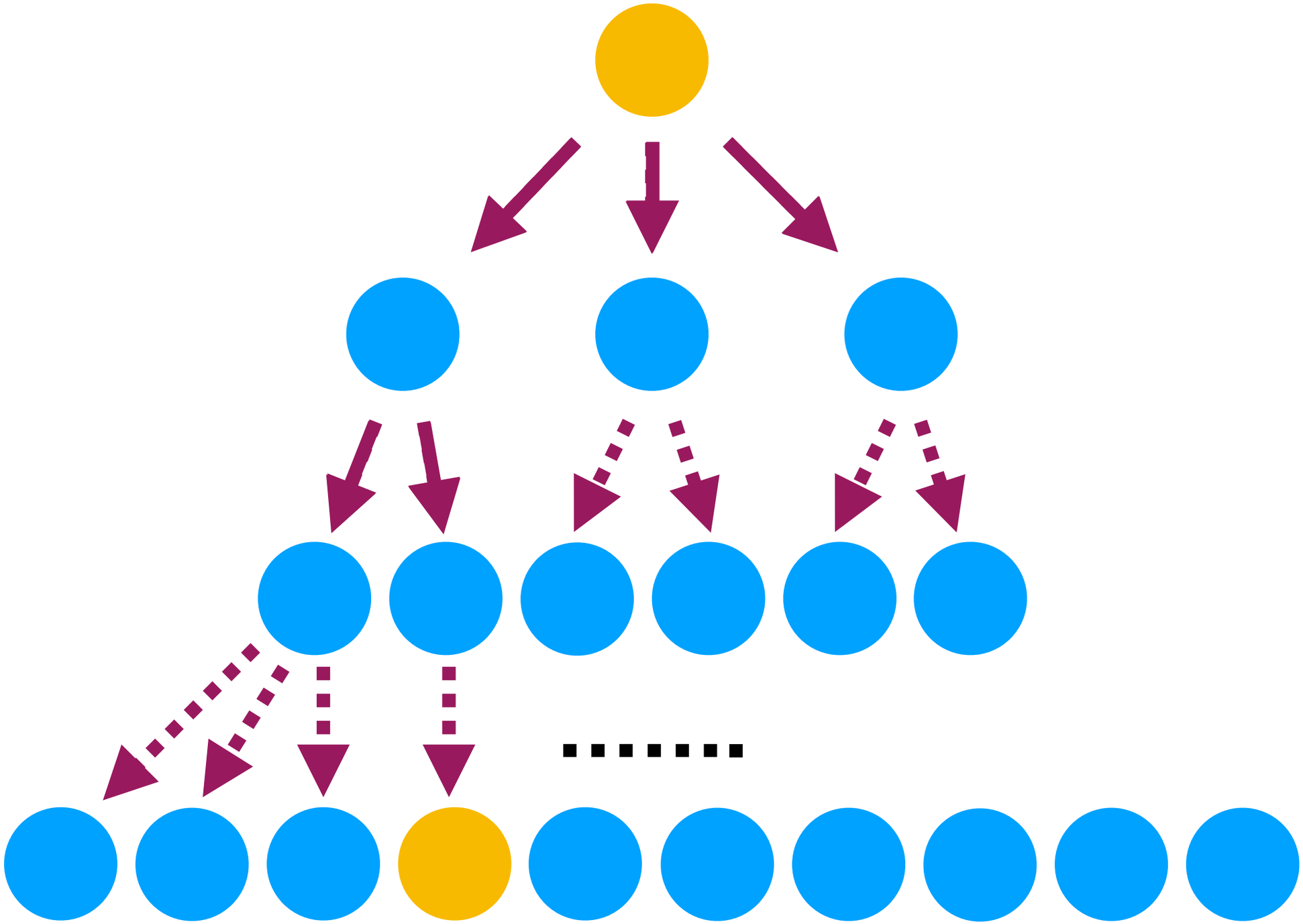
Breath-First Search (BFS)



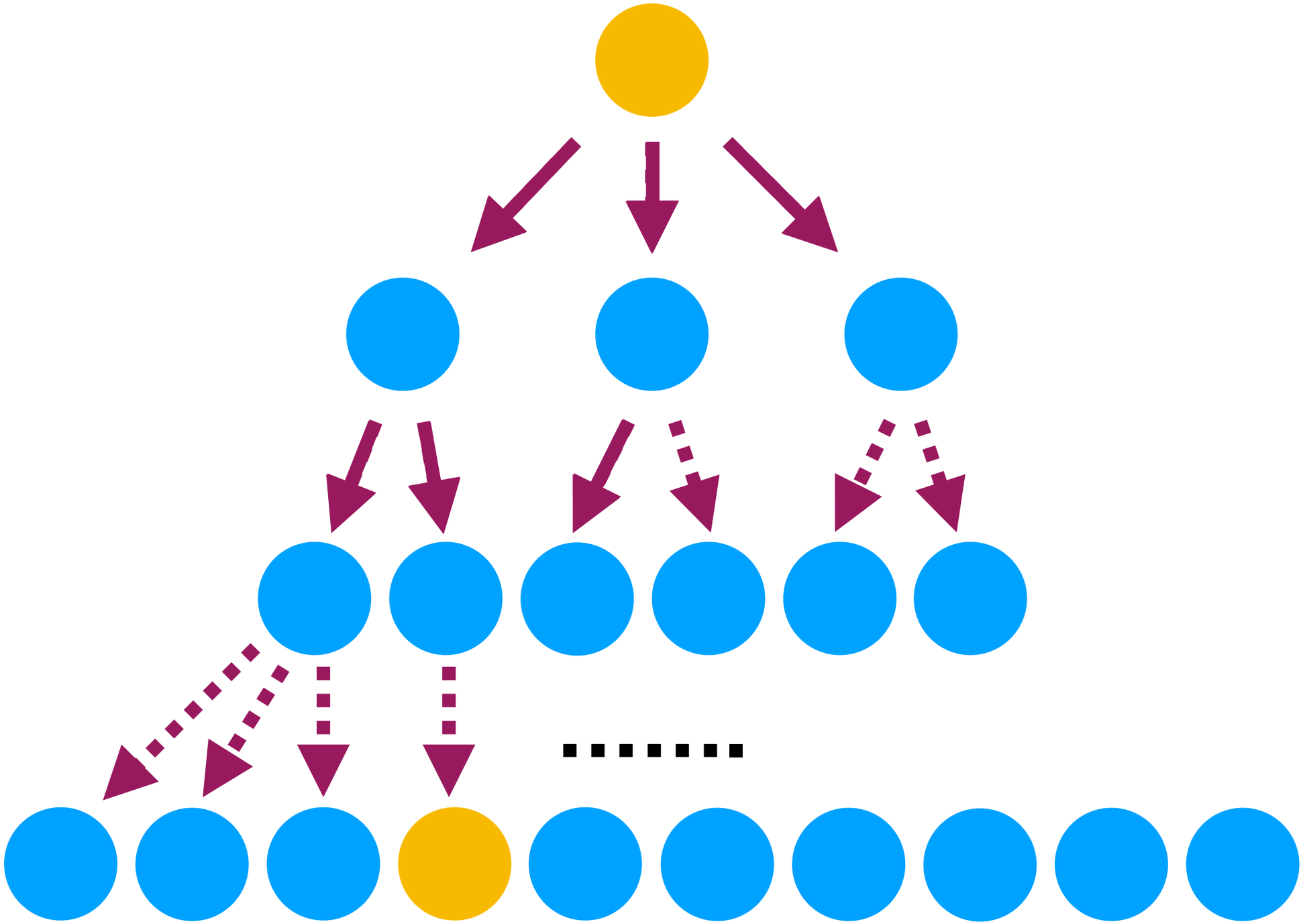
Breath-First Search (BFS)



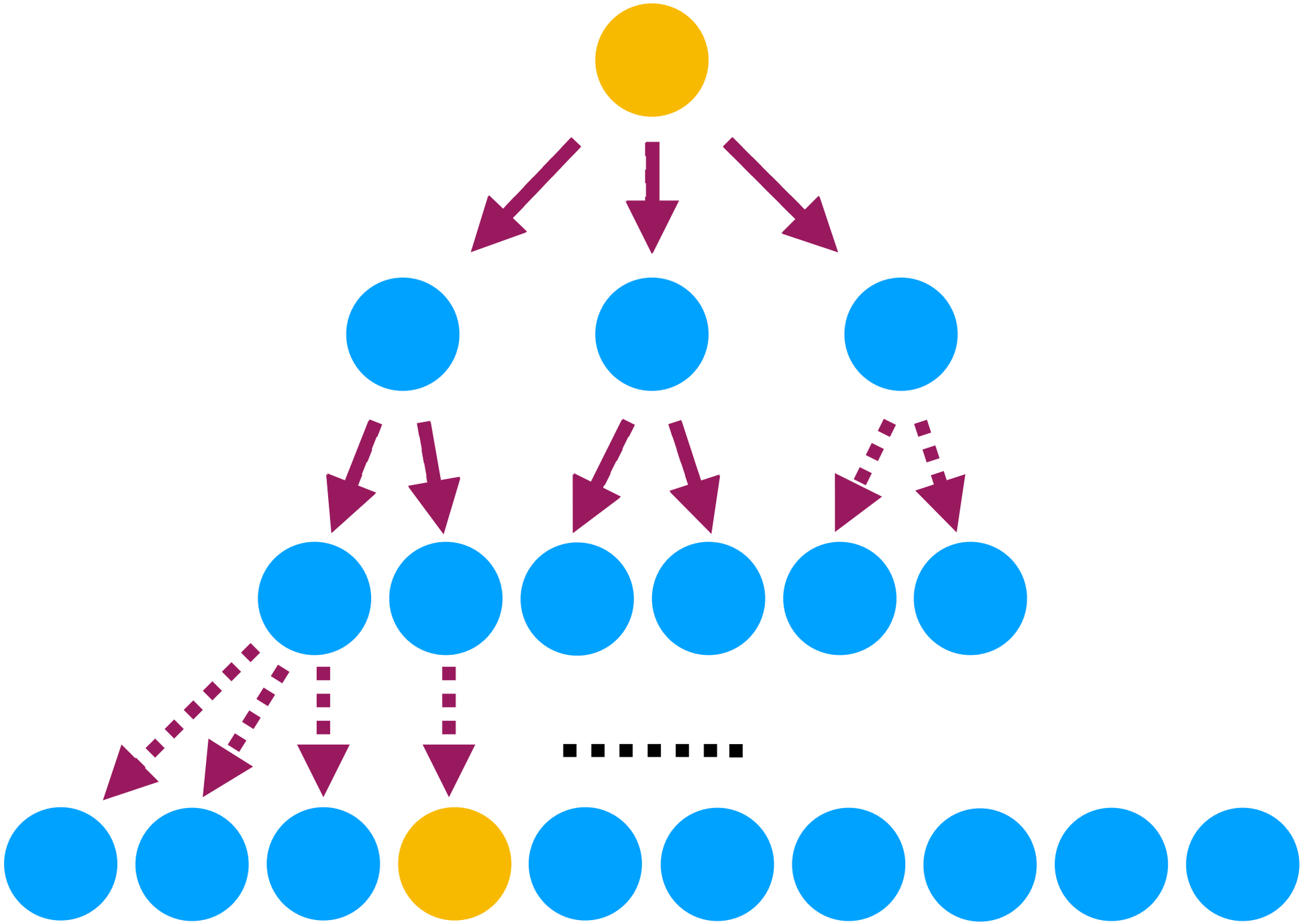
Breath-First Search (BFS)



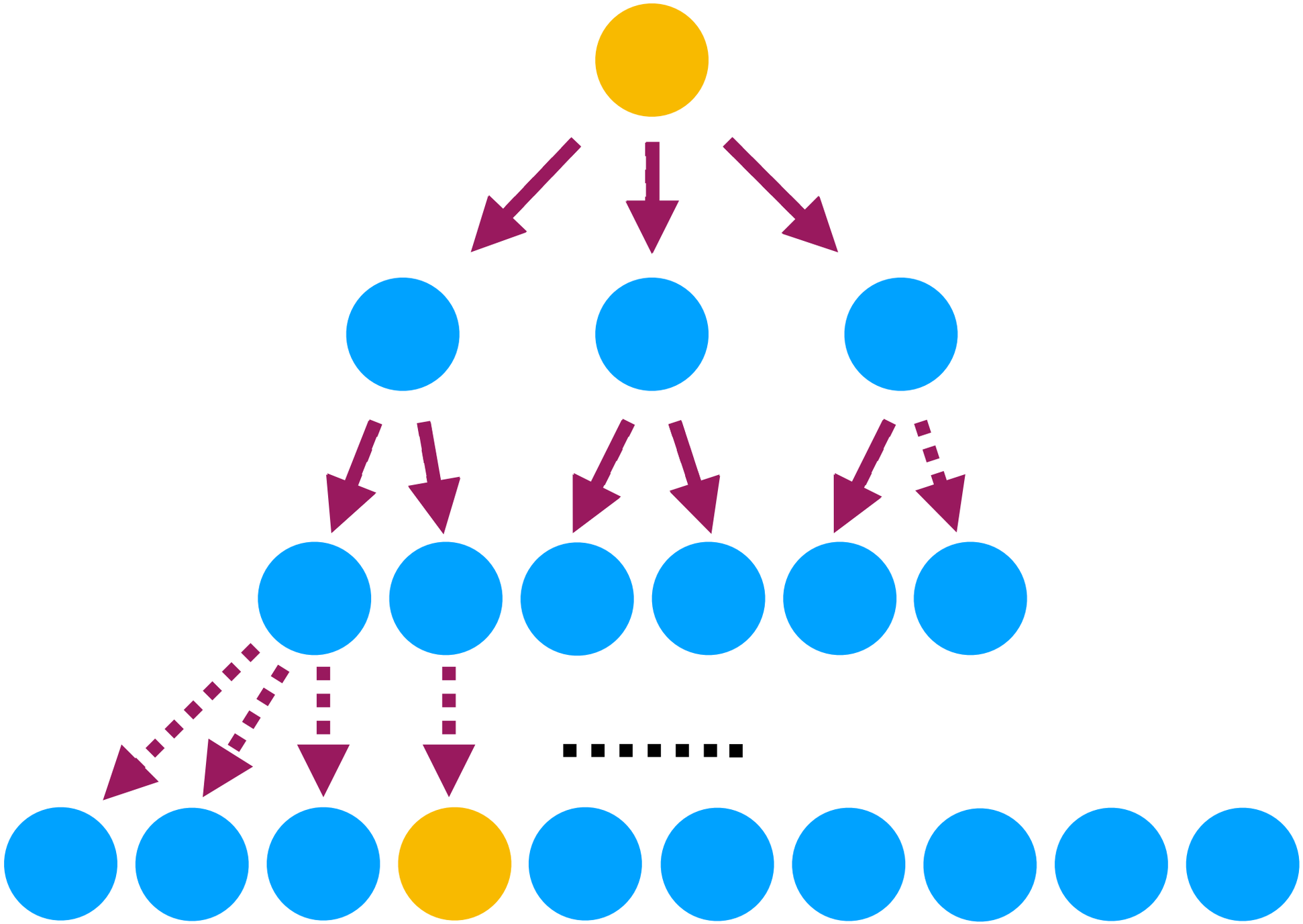
Breath-First Search (BFS)



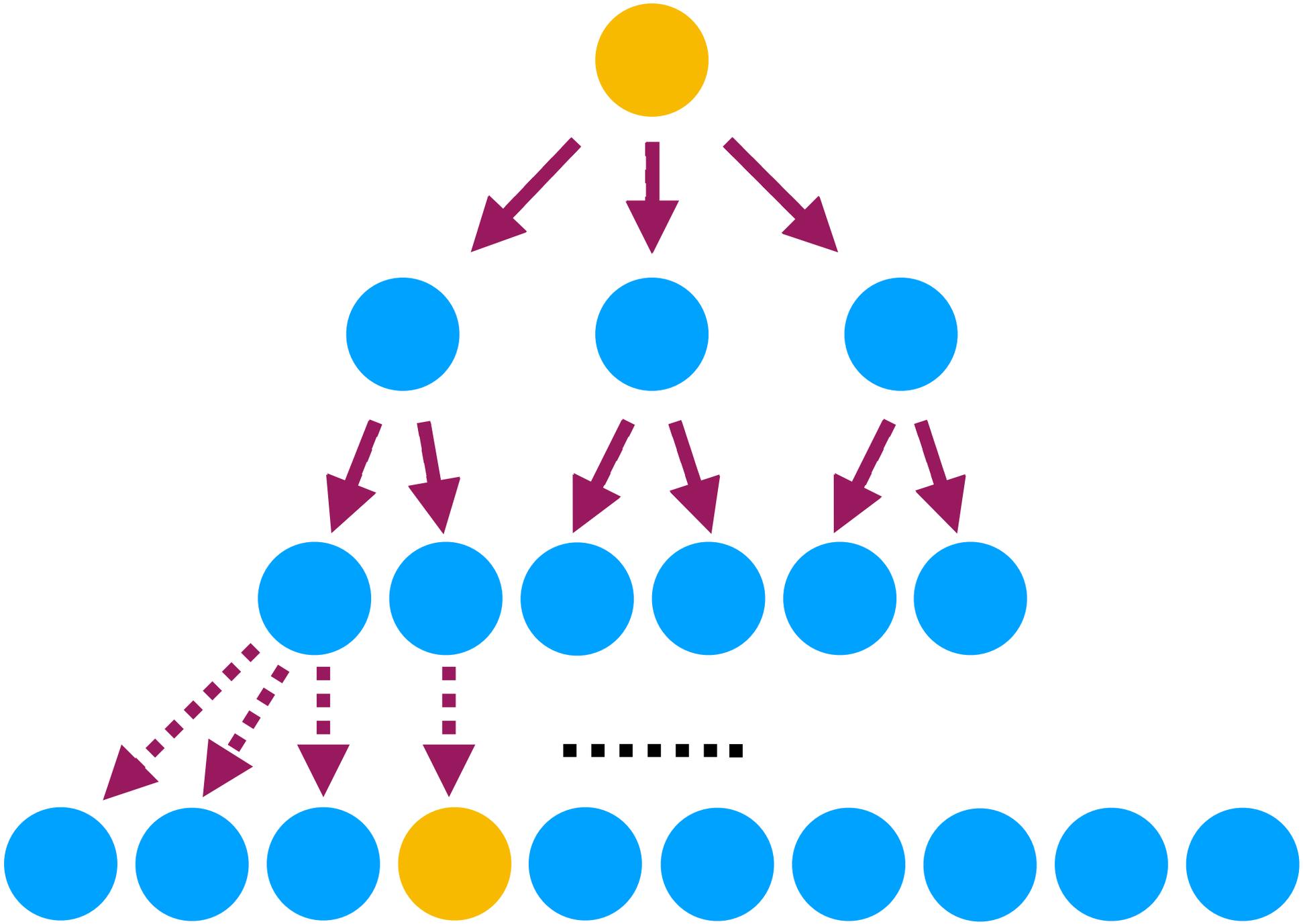
Breath-First Search (BFS)



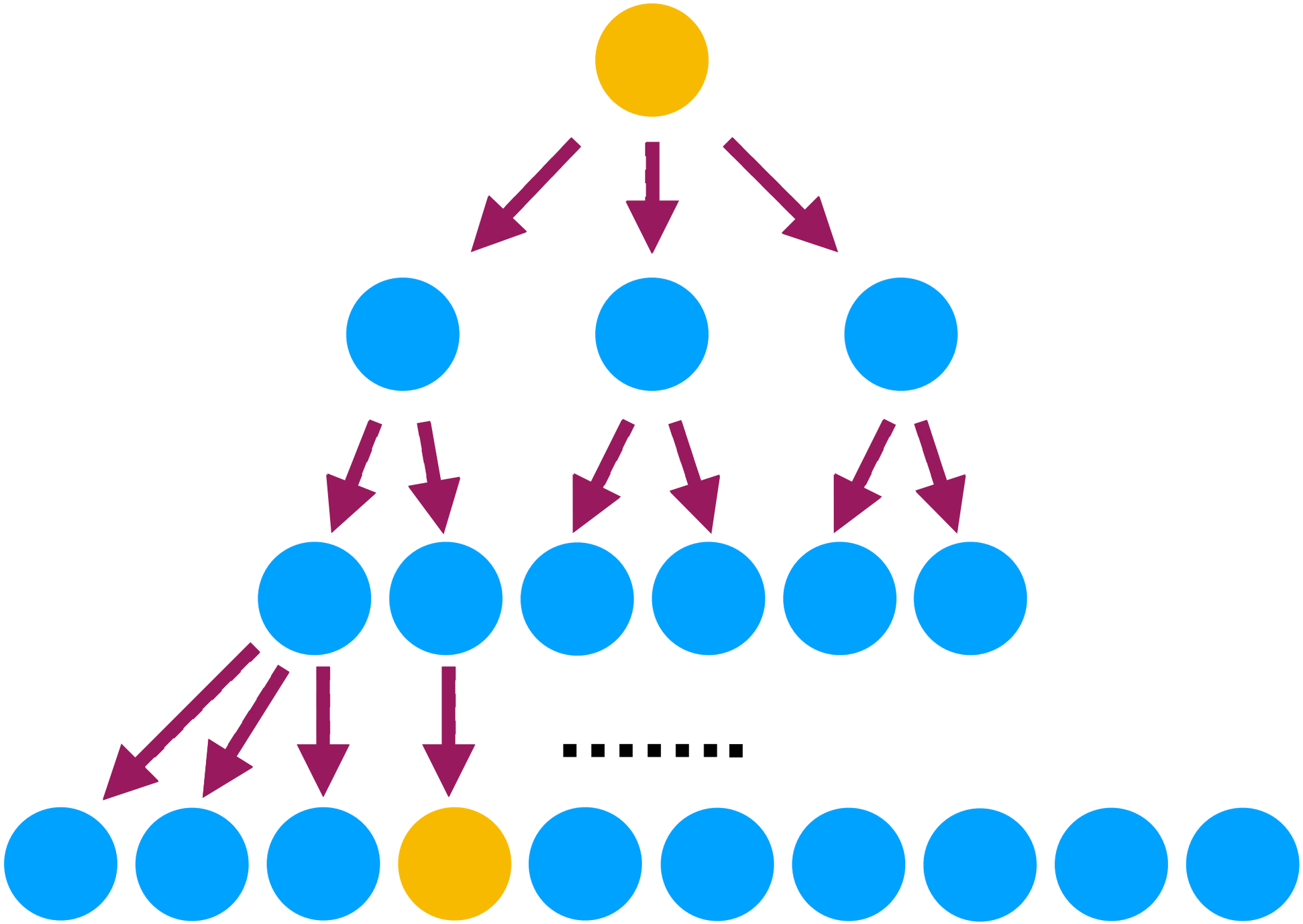
Breath-First Search (BFS)



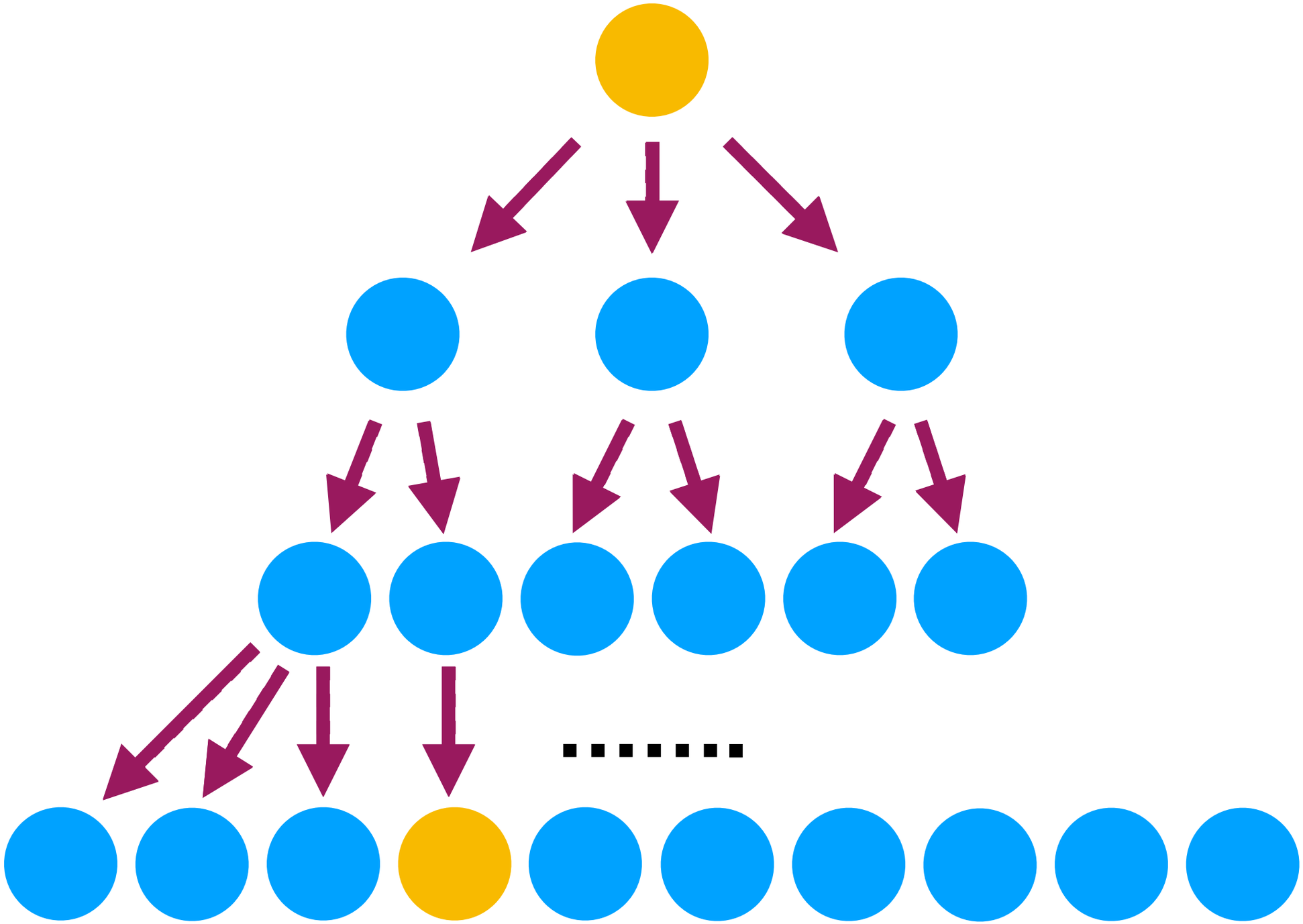
Breath-First Search (BFS)



Breath-First Search (BFS)



Breath-First Search (BFS)



Implemented with FIFO queue.

How Good is a Search Algorithm?

- Completeness:
  - Guarantee to find a solution if exists.
- Optimality:
  - Guarantee to find the least cost (largest utility) path.
- Time complexity
- Space complexity

How Good is a Search Algorithm?

- Completeness:
  - Guarantee to find a solution if exists.
- Optimality:
  - Guarantee to find the least cost (largest utility) path.

caution: not about complexity

- Time complexity
- Space complexity

How Good is BFS?

- Completeness:
  - Yes

b : the maximum #edge on a node
d : the maximum depth of the search tree

- Optimality:
  - Yes when costs on all edges are equal

- Time complexity
  - $O(b^d)$

- Space complexity
  - $O(b^d)$

How Good is BFS?

- Completeness:
  - Yes

b : the maximum #edge on a node
d : the maximum depth of the search tree

- Optimality:
  - Yes when costs on all edges are equal

- Time complexity
  - $O(b^d)$

Difficult to improve if no more information about the search tree is given

- Space complexity
  - $O(b^d)$

How Good is BFS?

- Completeness:
  - Yes

b : the maximum #edge on a node
d : the maximum depth of the search tree

- Optimality:
  - Yes when costs on all edges are equal
- Time complexity
  - $O(b^d)$

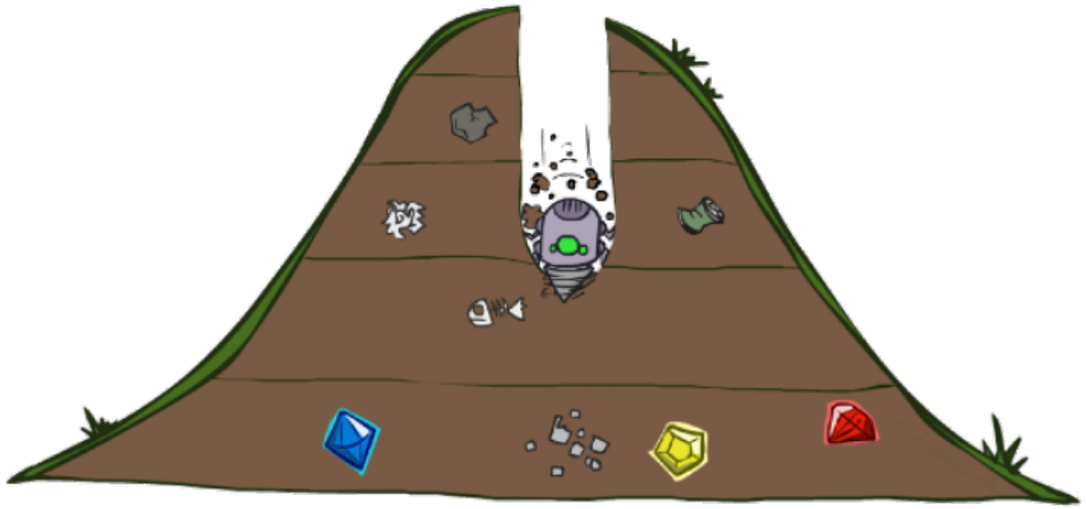
Difficult to improve if no more information about the search tree is given

- Space complexity
  - $O(b^d)$

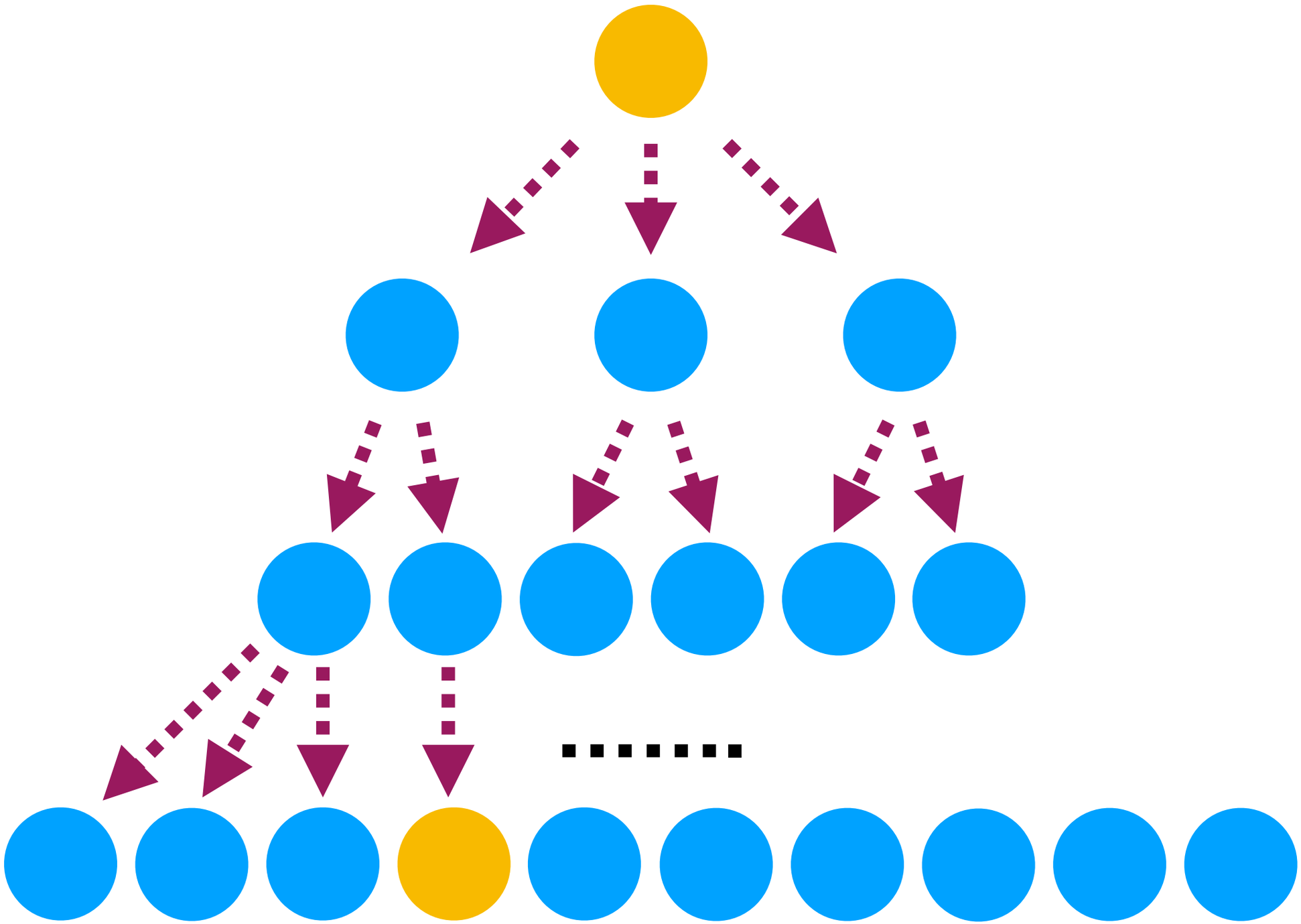
Big issue for real-world application!
Anyway to improve?

Outline: Decision Making (I)

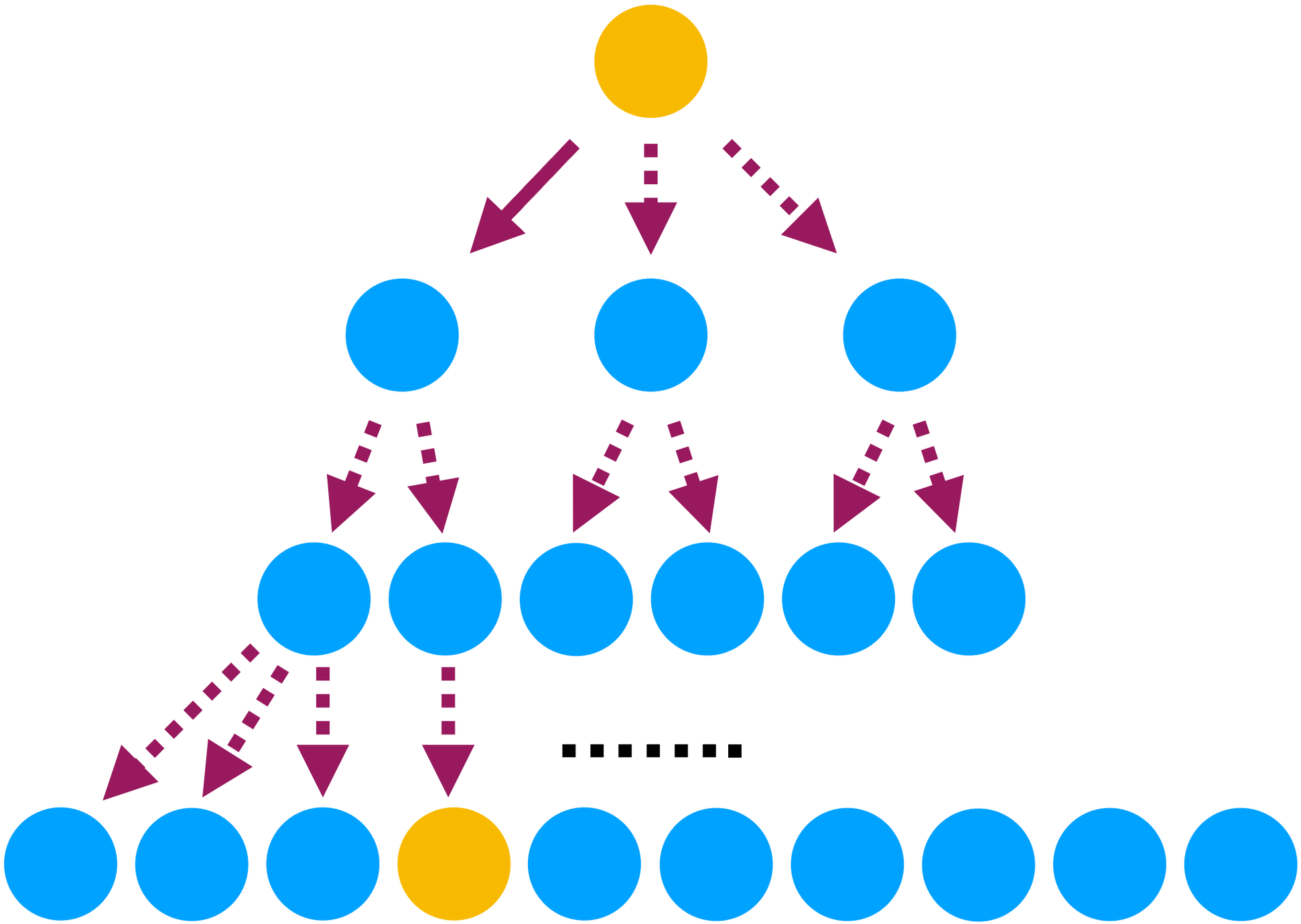
- Uninformed search
  - Breadth-first search
  - Depth-first search
- Informed search
  - Best-first search
  - A* search
- Take-Home Messages



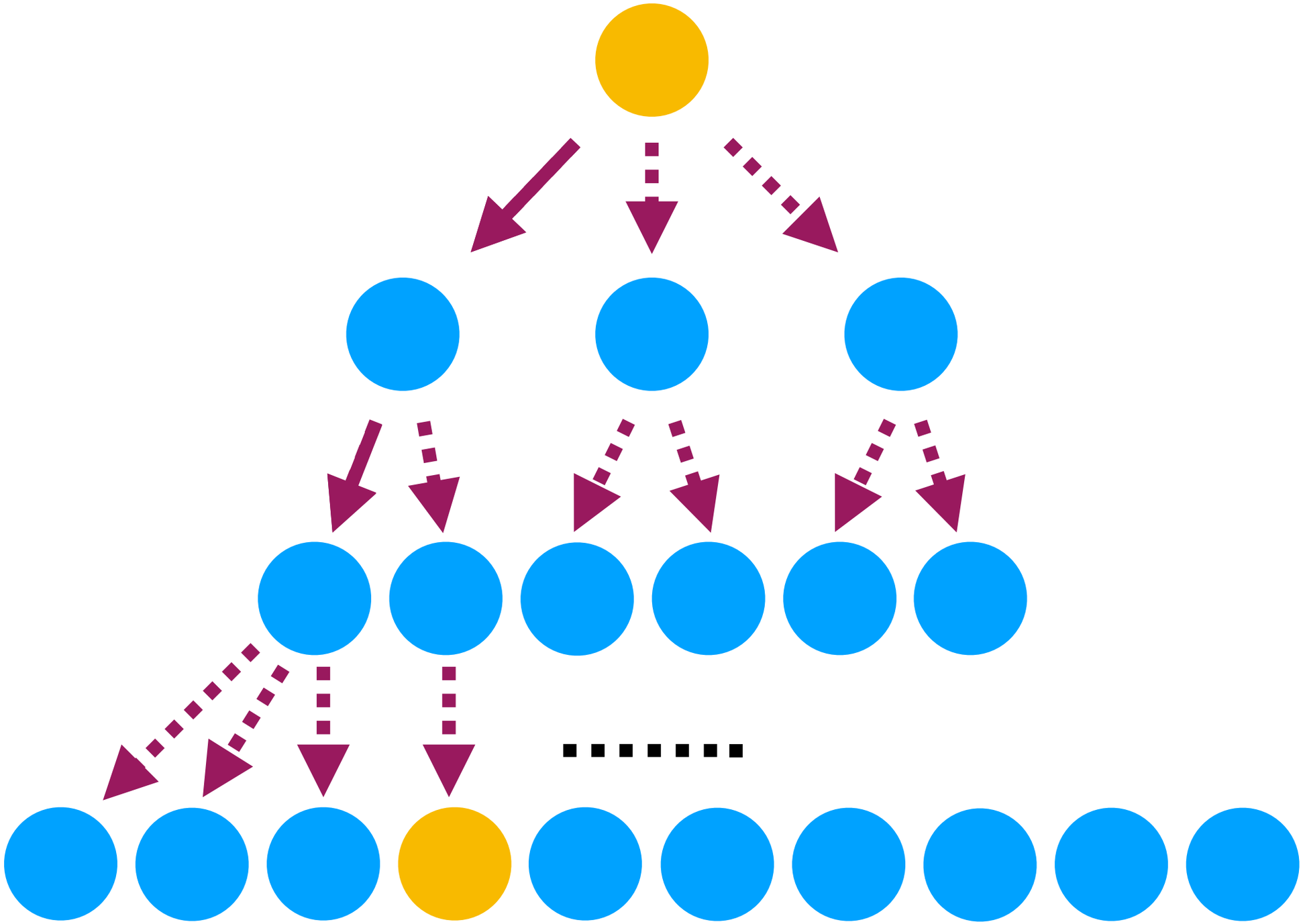
Depth-First Search (DFS)



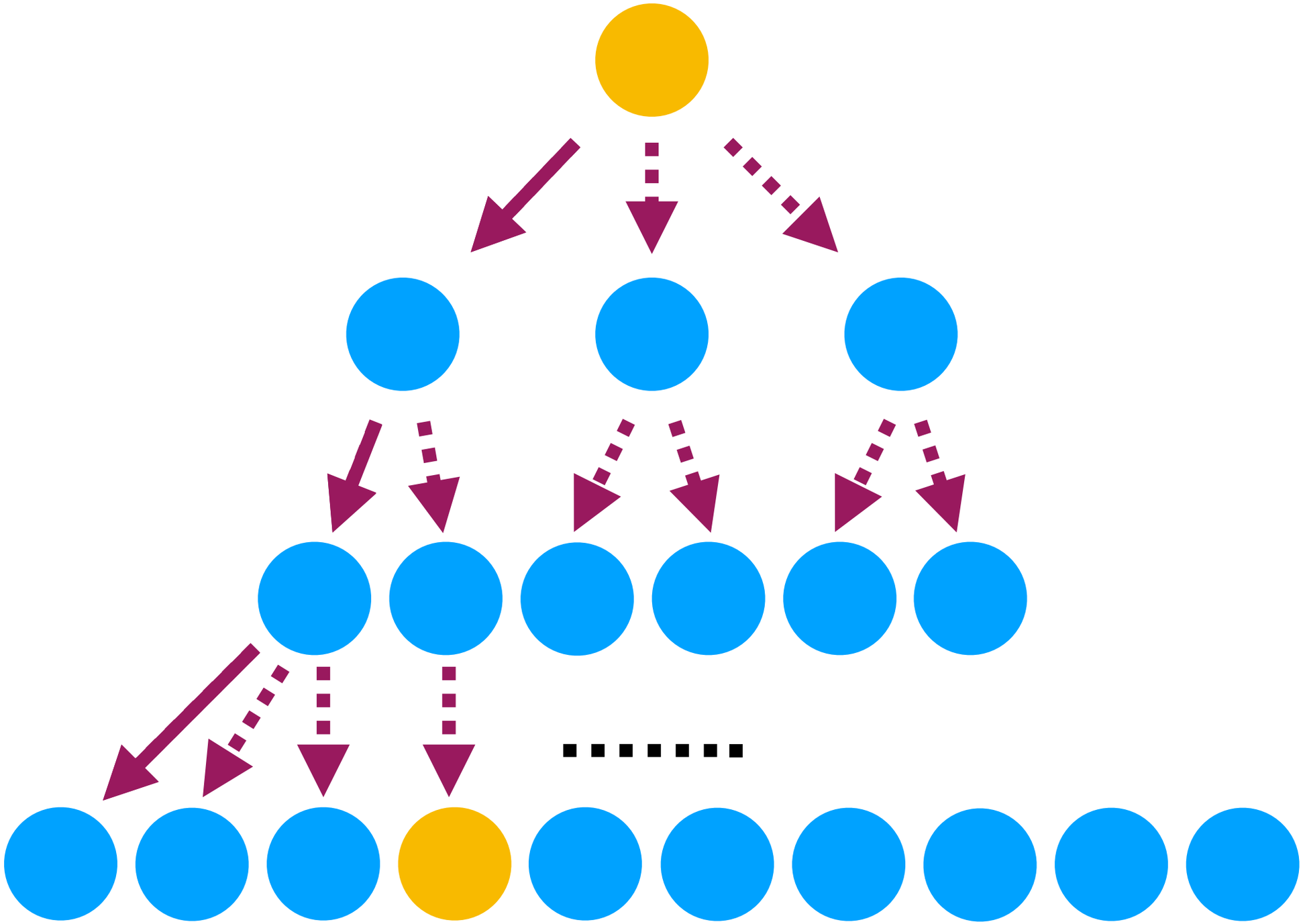
Depth-First Search (DFS)



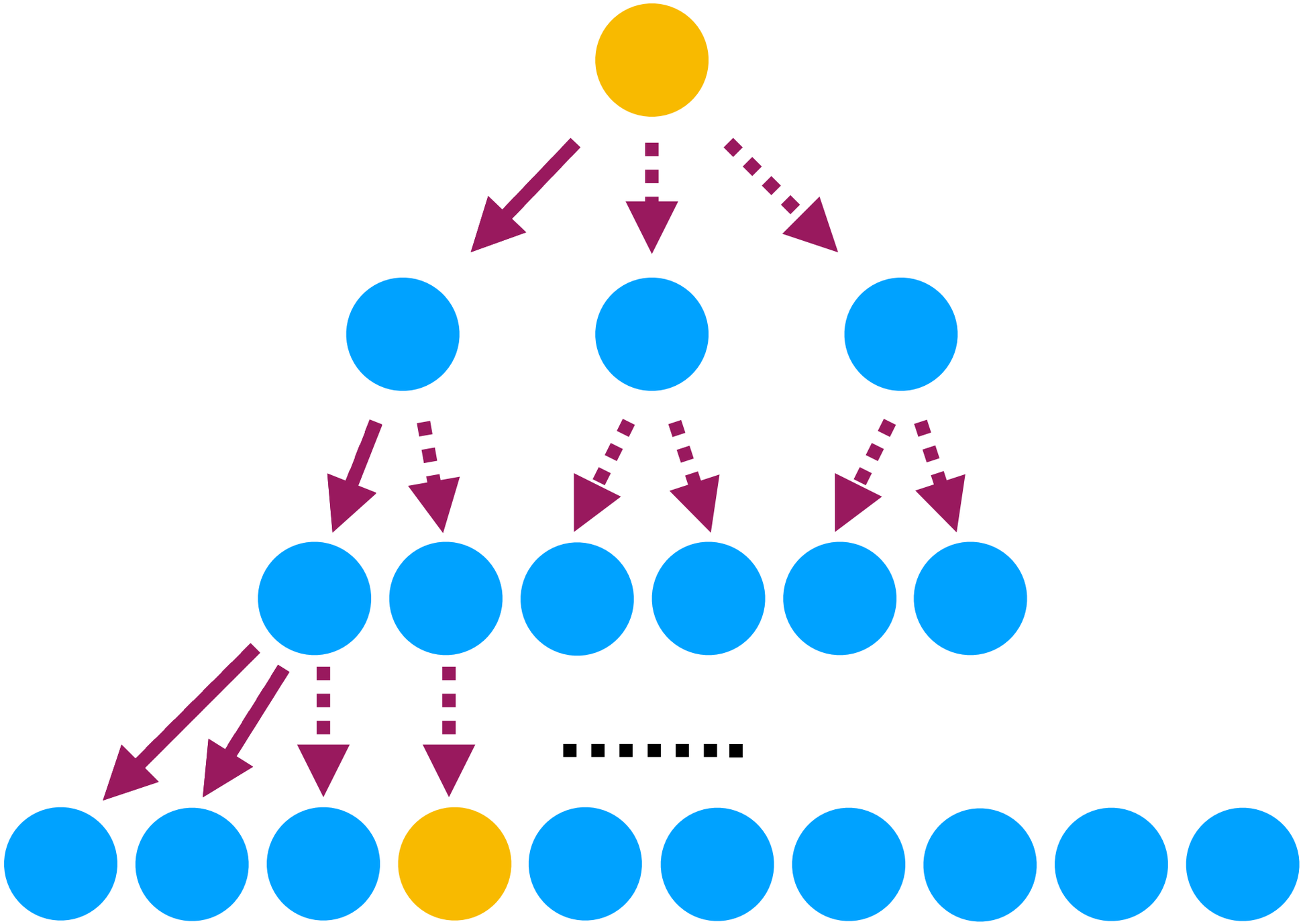
Depth-First Search (DFS)



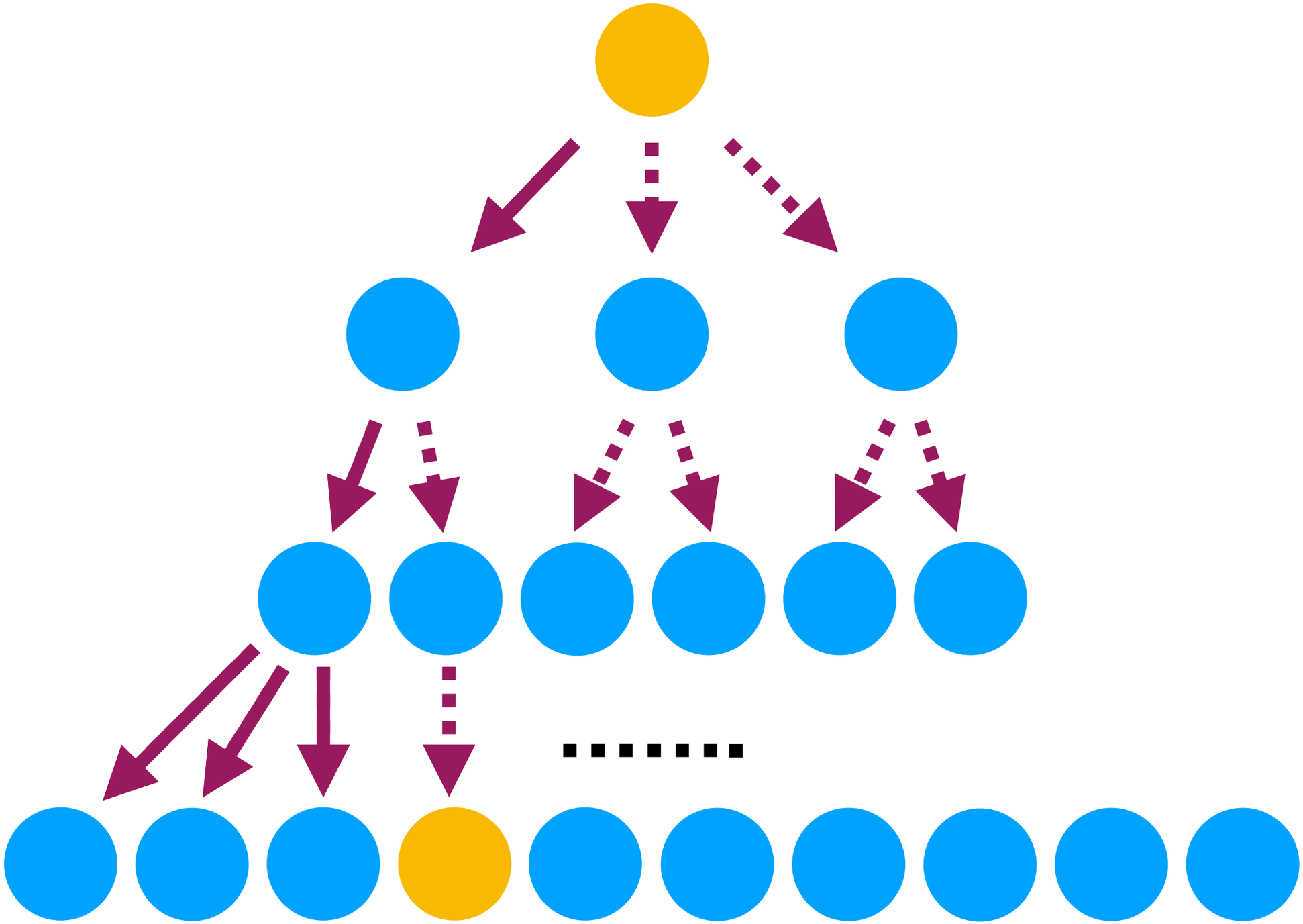
Depth-First Search (DFS)



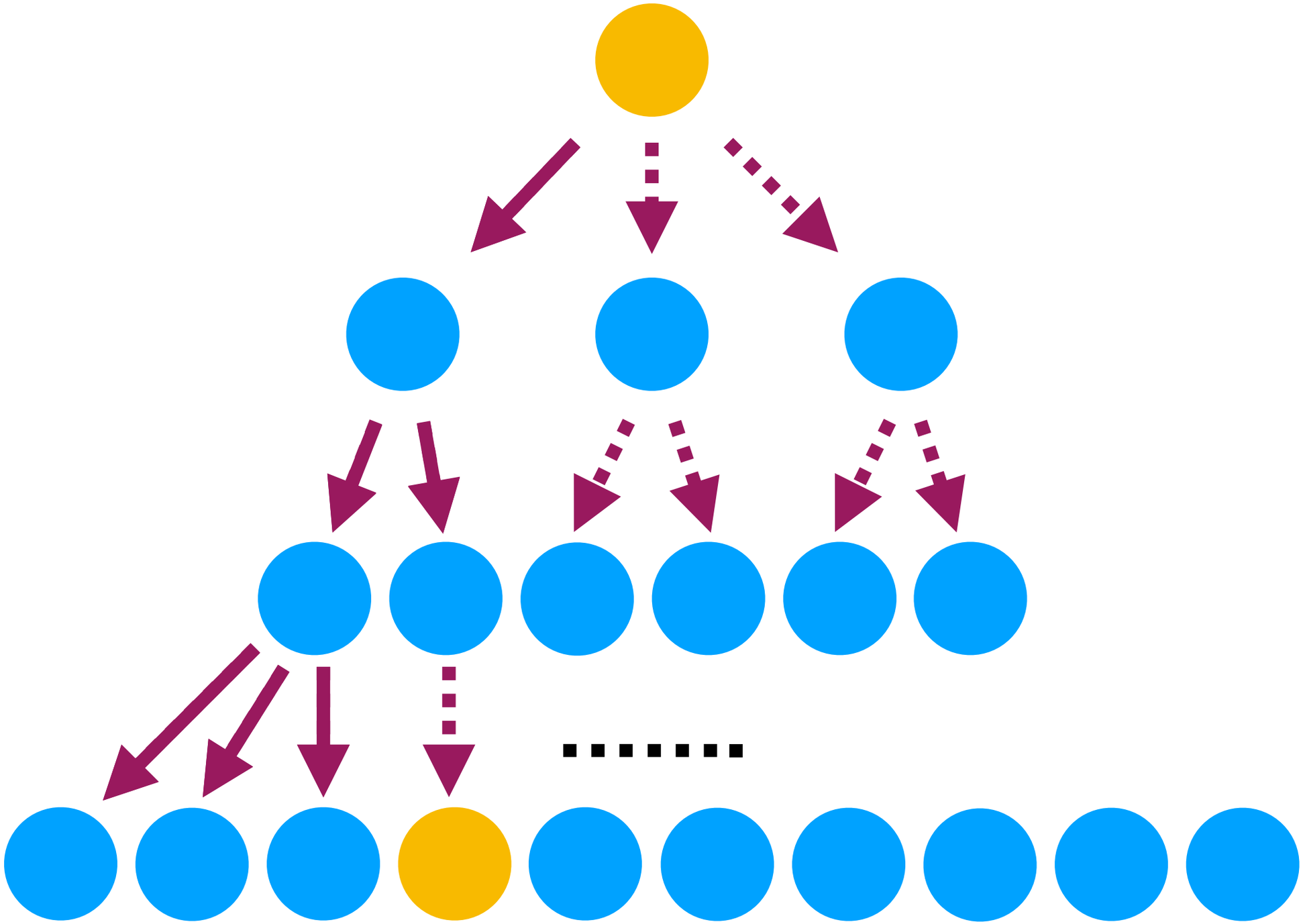
Depth-First Search (DFS)



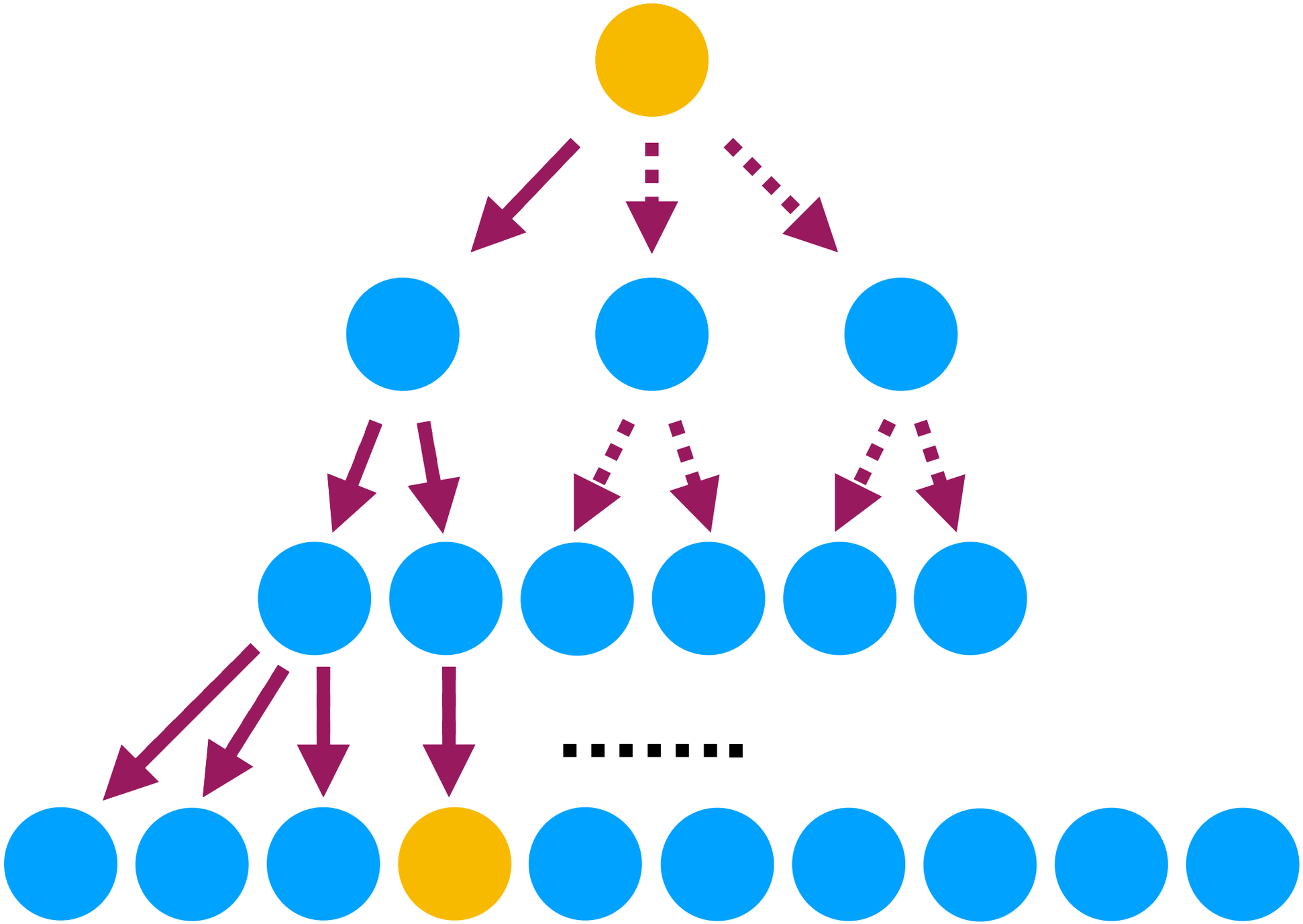
Depth-First Search (DFS)



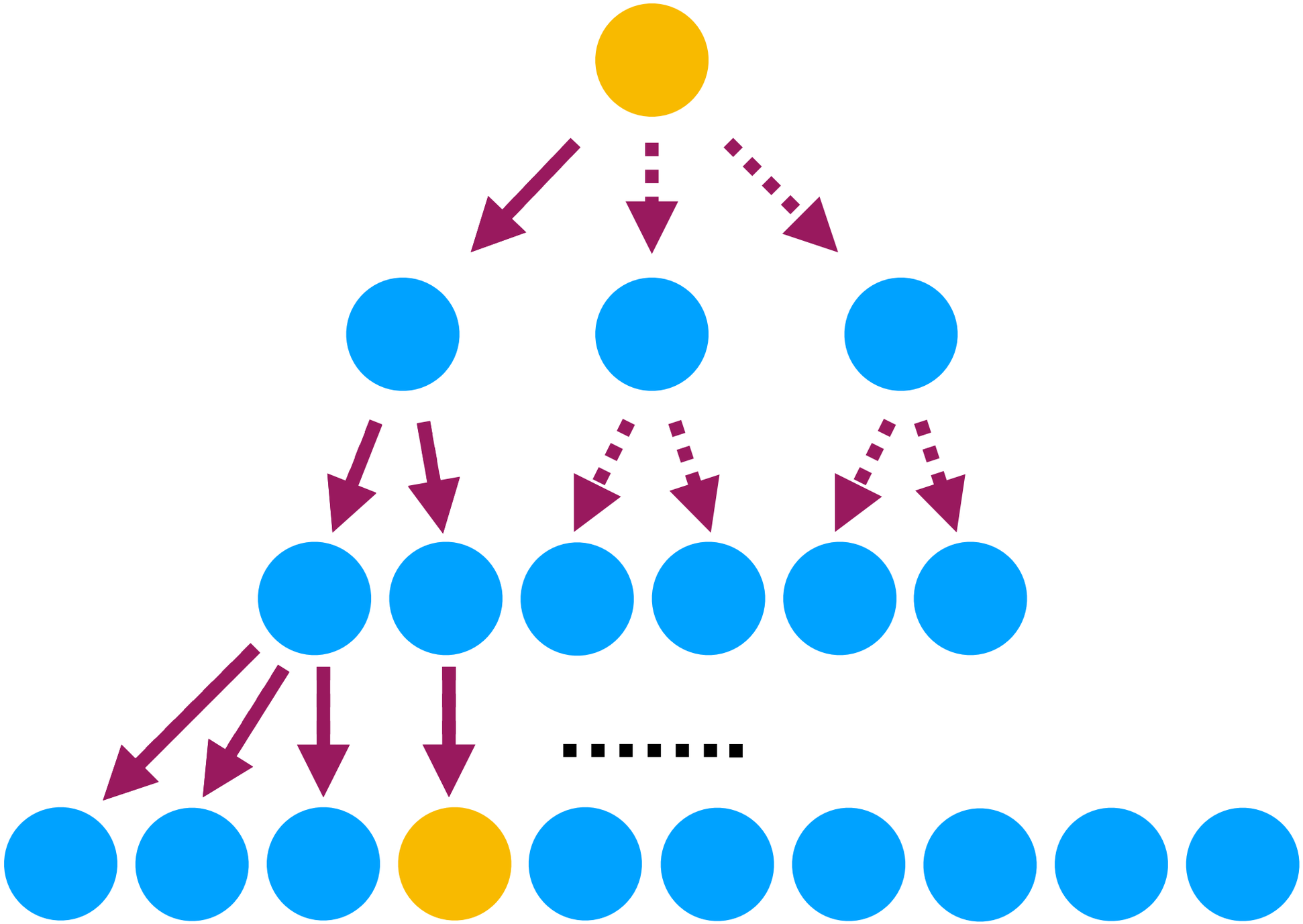
Depth-First Search (DFS)



Depth-First Search (DFS)



Depth-First Search (DFS)



Implemented with LIFO queue (stack).

How Good is DFS?

- Completeness:
  - NO! (why?)
- Optimality:
  - NO! (why?)
- Time complexity
  - $O(b^d)$
- Space complexity
  - $O(bd)$

b : the maximum #edge on a node
d : the maximum depth of the search tree

How Good is DFS?

- Completeness:
  - NO! (why?)

b : the maximum #edge on a node
d : the maximum depth of the search tree

- Optimality:
  - NO! (why?)

- Time complexity
  - $O(b^d)$

- Space complexity
  - $O(bd)$

Much more reasonable than BFS.
Making DFS more widely used in practice.

BFS vs. DFS

Can we get the benefits of both BFS and DFS?
Can we deal with non-uniform cost (utility) problems?

Iterative Deepening Search

- Run a DFS with depth limit 1. If no solution…
- Run a DFS with depth limit 2. If no solution…
- Run a DFS with depth limit 3. …

Iterative Deepening Search

- Run a DFS with depth limit 1. If no solution…
- Run a DFS with depth limit 2. If no solution…
- Run a DFS with depth limit 3. …

Why can we do this? Aren't many nodes searched repeatedly?

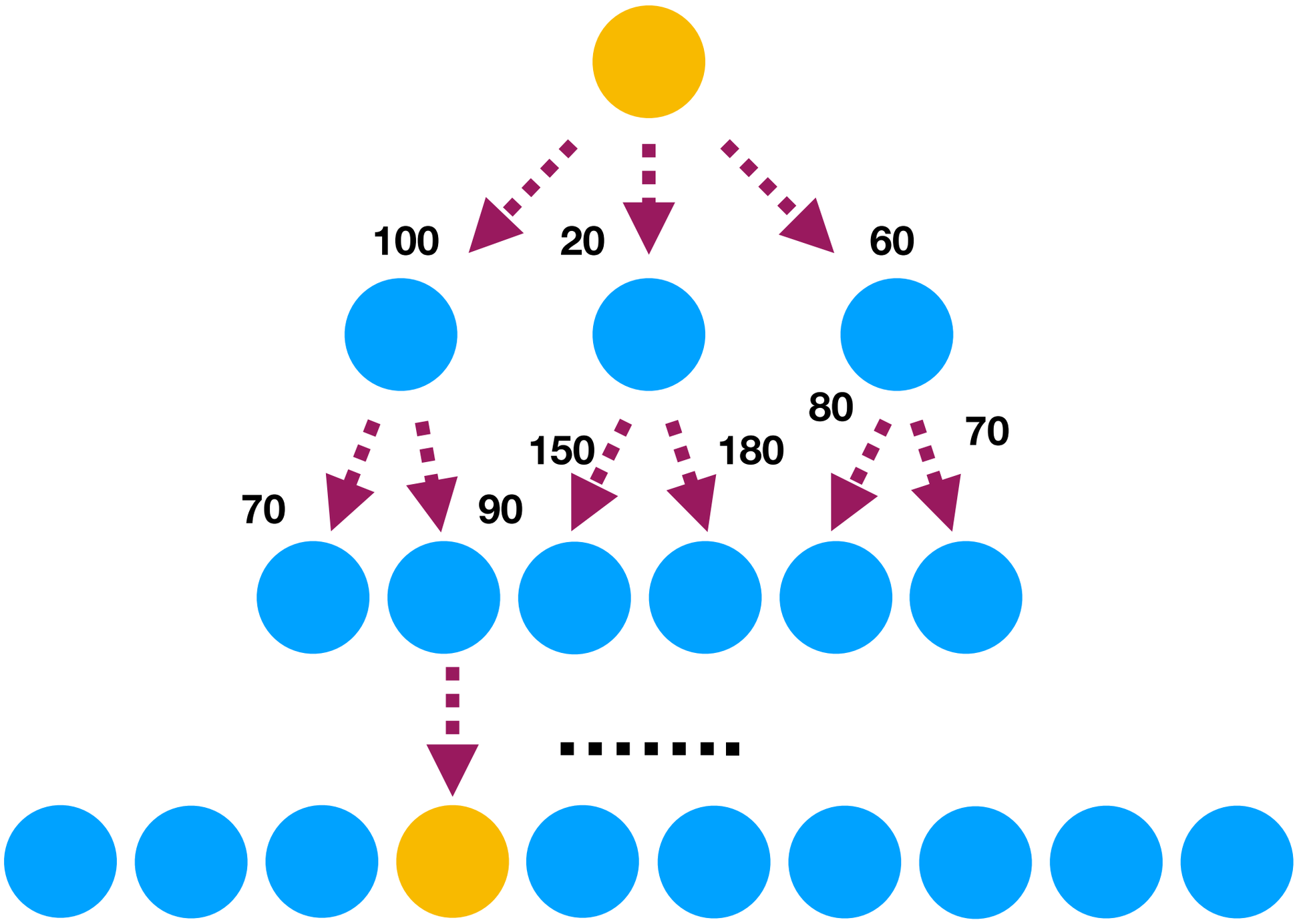
Iterative Deepening Search

- Run a DFS with depth limit 1. If no solution…
- Run a DFS with depth limit 2. If no solution…
- Run a DFS with depth limit 3. …

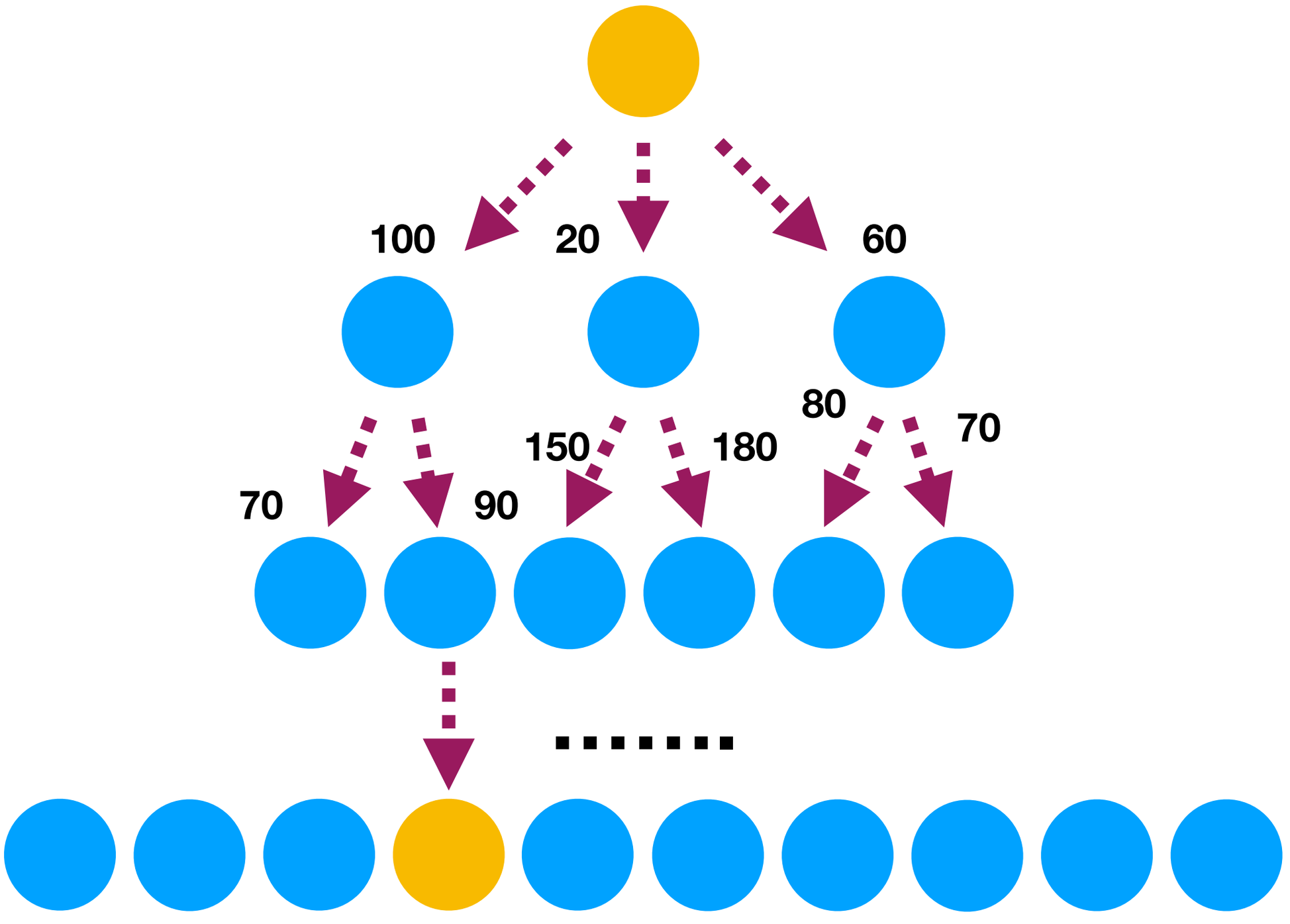
Why can we do this? Aren't many nodes searched repeatedly?

Actually, the time complexity is dominated by the latest DFS!

Cost-Sensitive Search

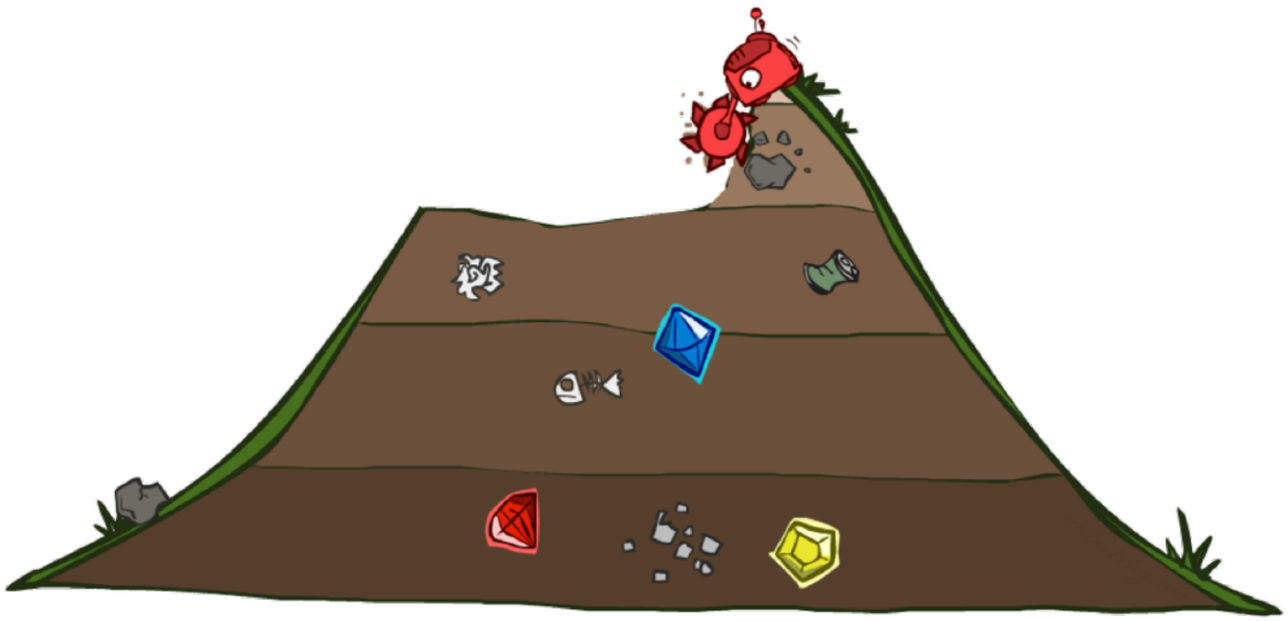


Cost-Sensitive Search

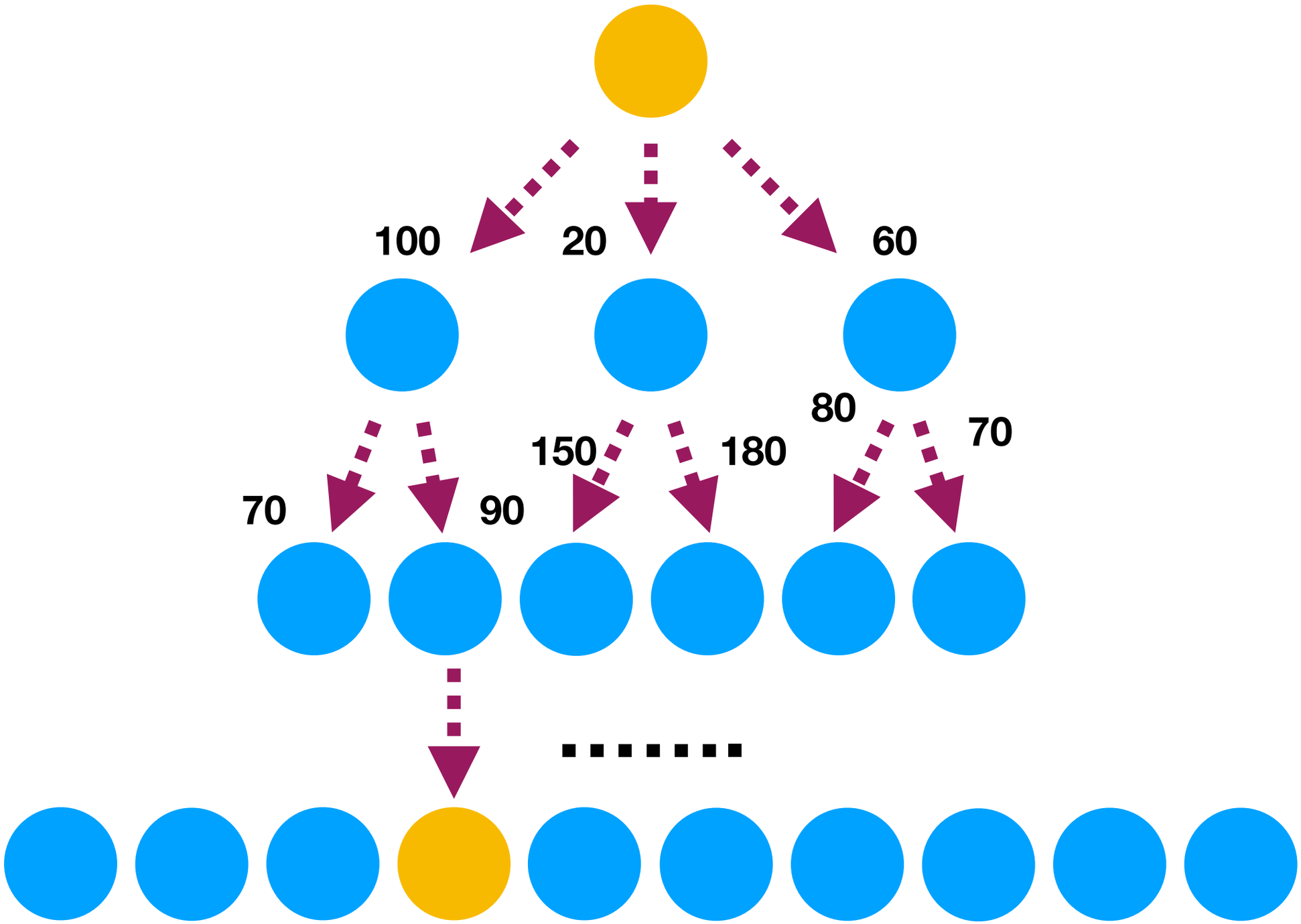


Neither BFS nor DFS works.

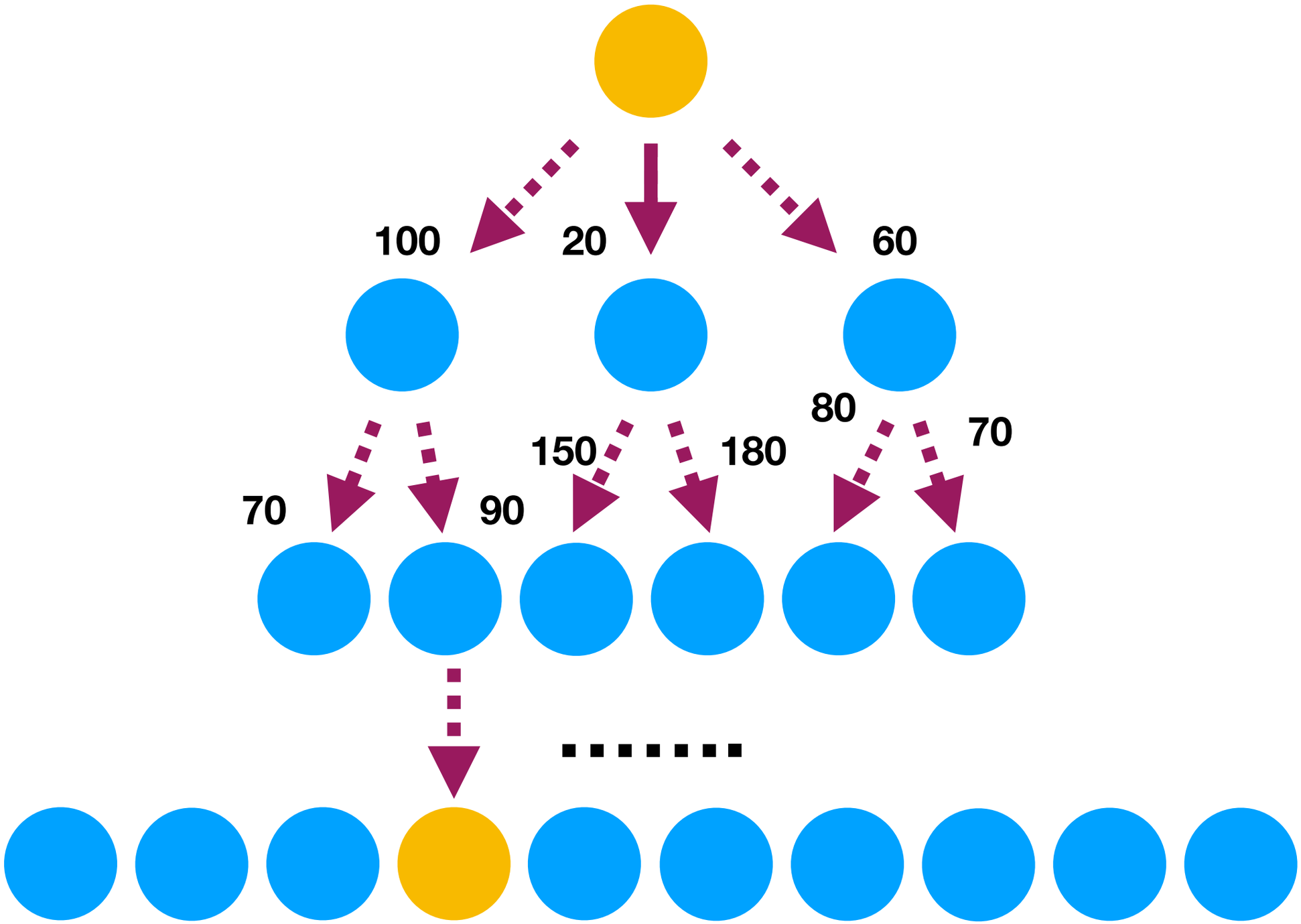
Uniform Cost Search (UCS)



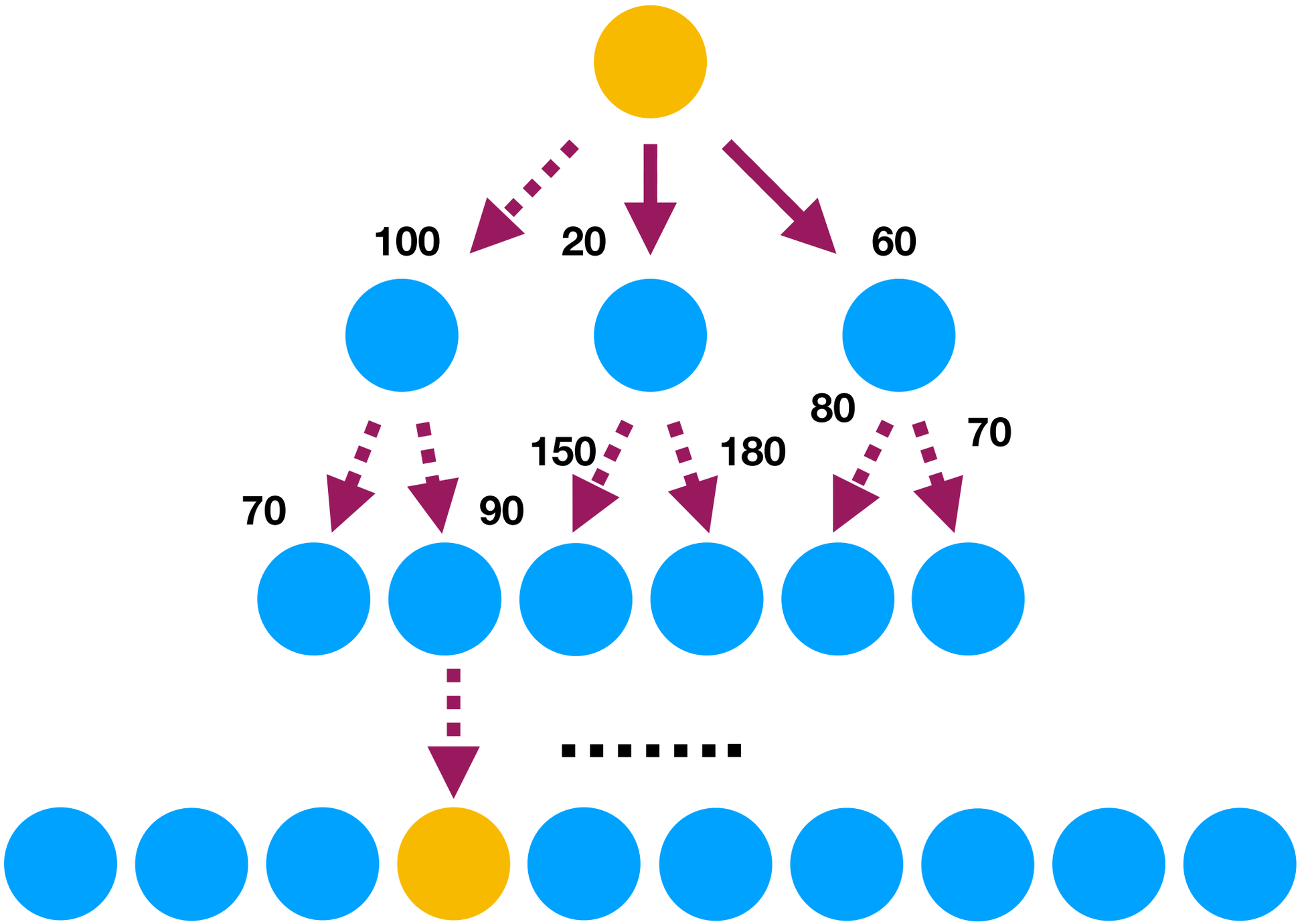
Uniform Cost Search (UCS)



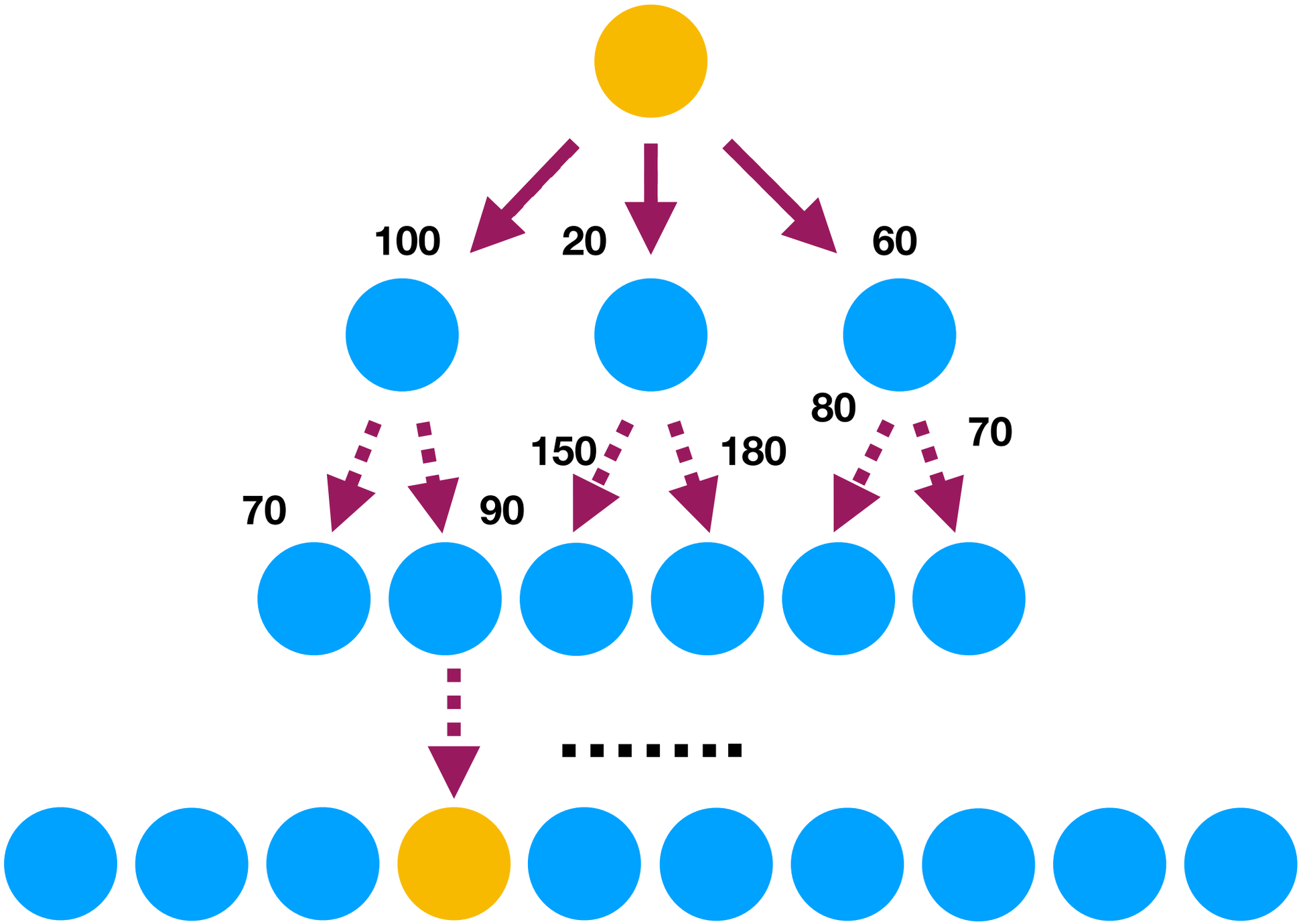
Uniform Cost Search (UCS)



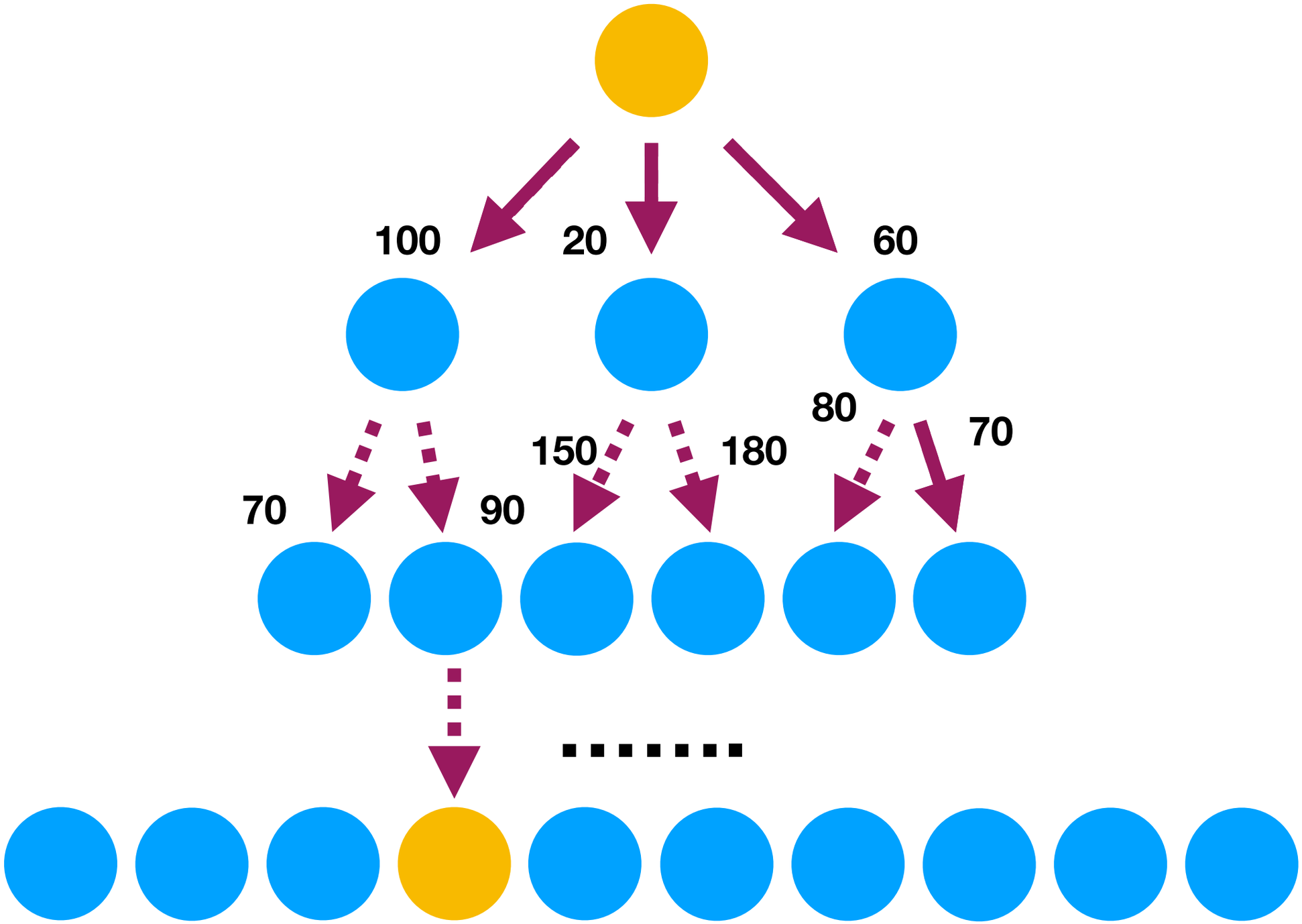
Uniform Cost Search (UCS)



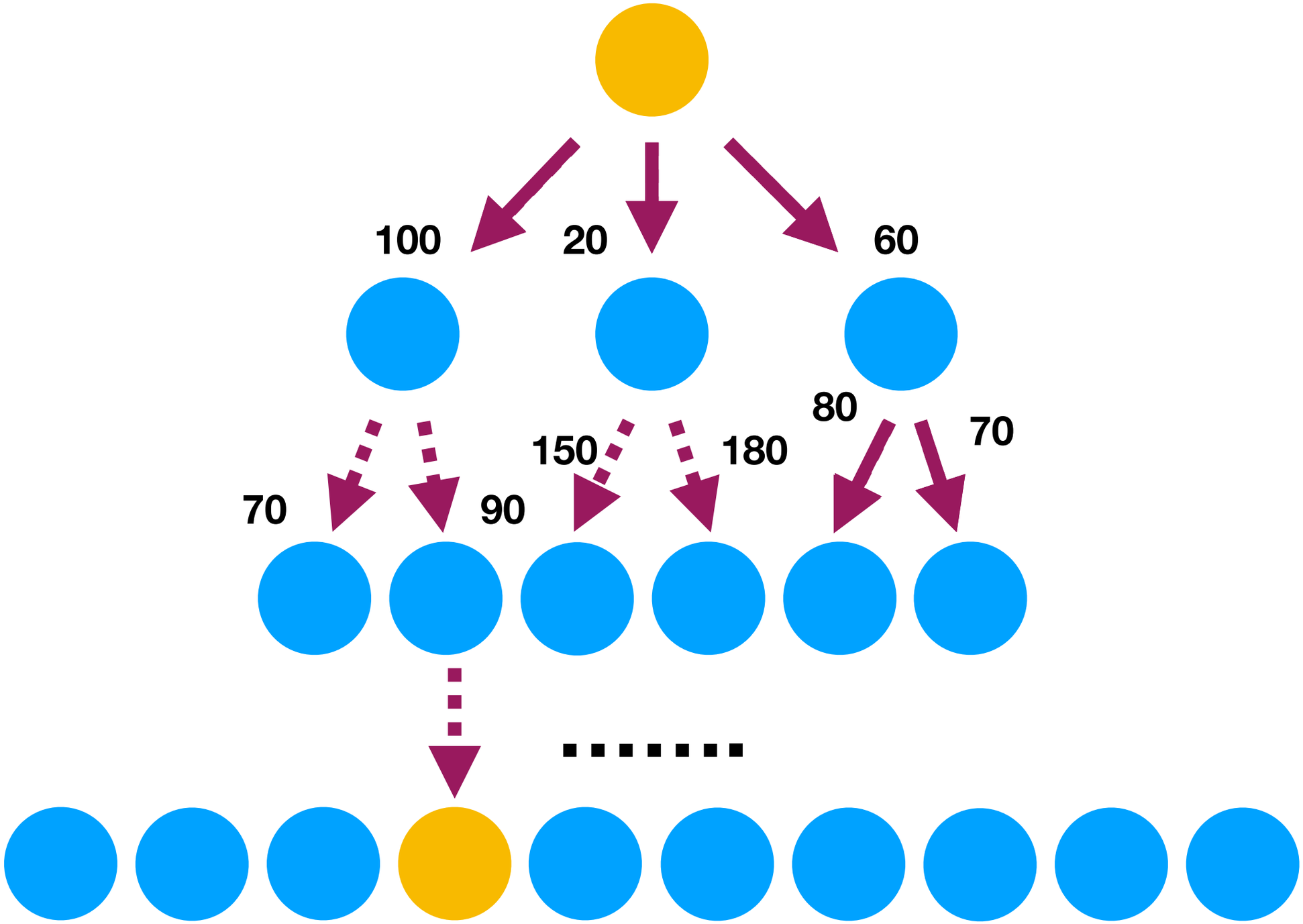
Uniform Cost Search (UCS)



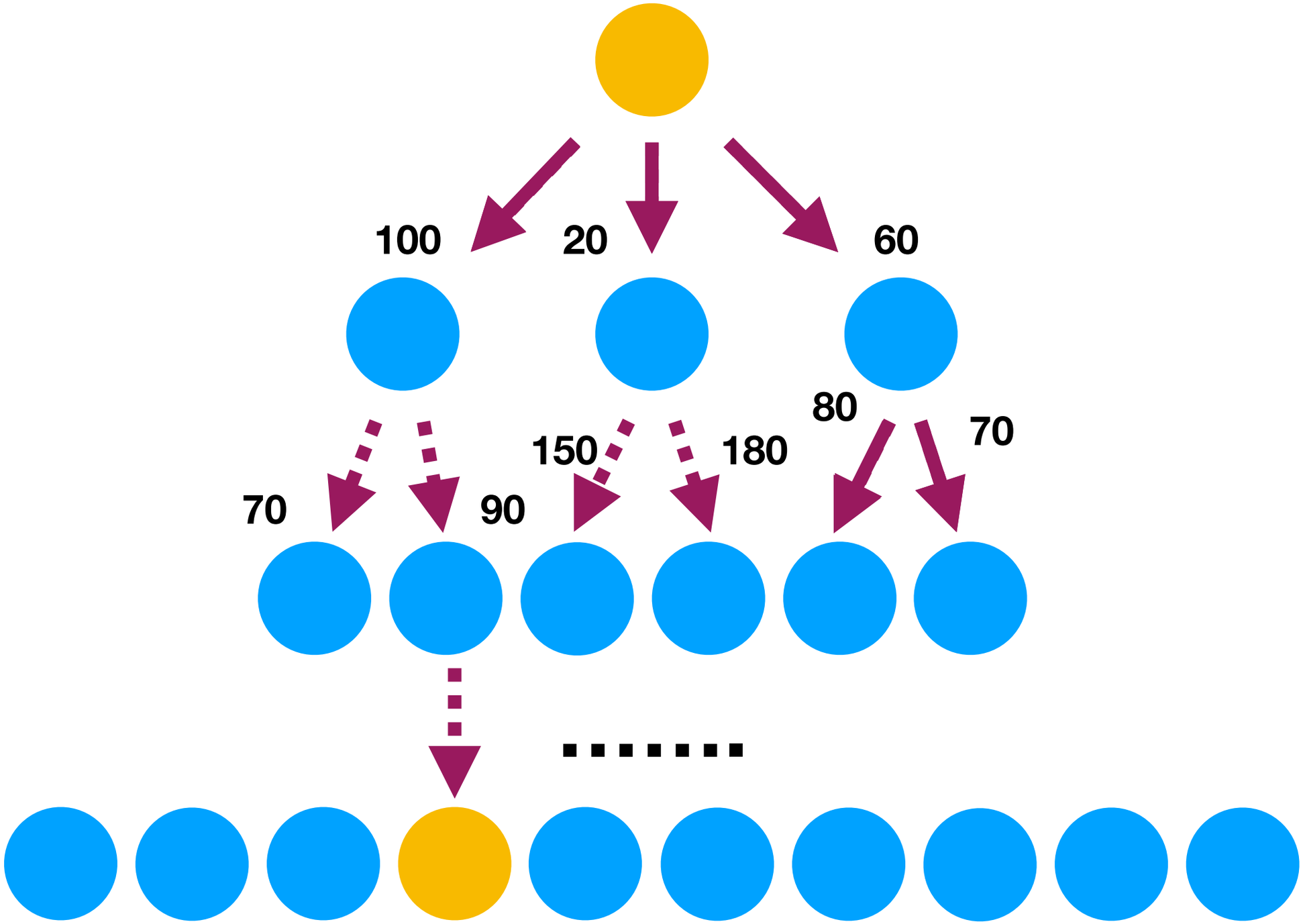
Uniform Cost Search (UCS)



Uniform Cost Search (UCS)



Uniform Cost Search (UCS)

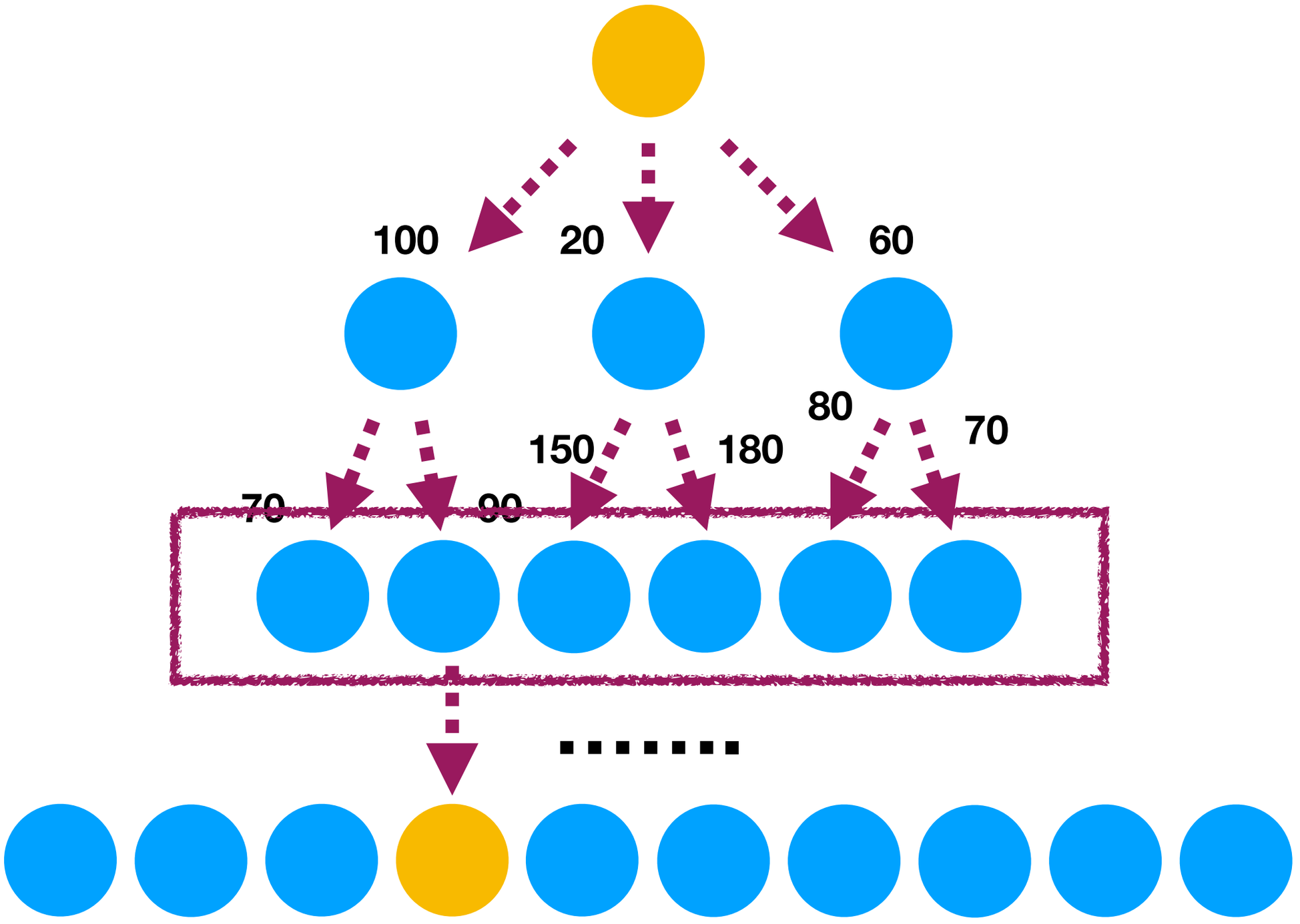


Implemented by priority queue (priority on cumulative cost).
Special case of the A* search.

Outline: Decision Making (I)

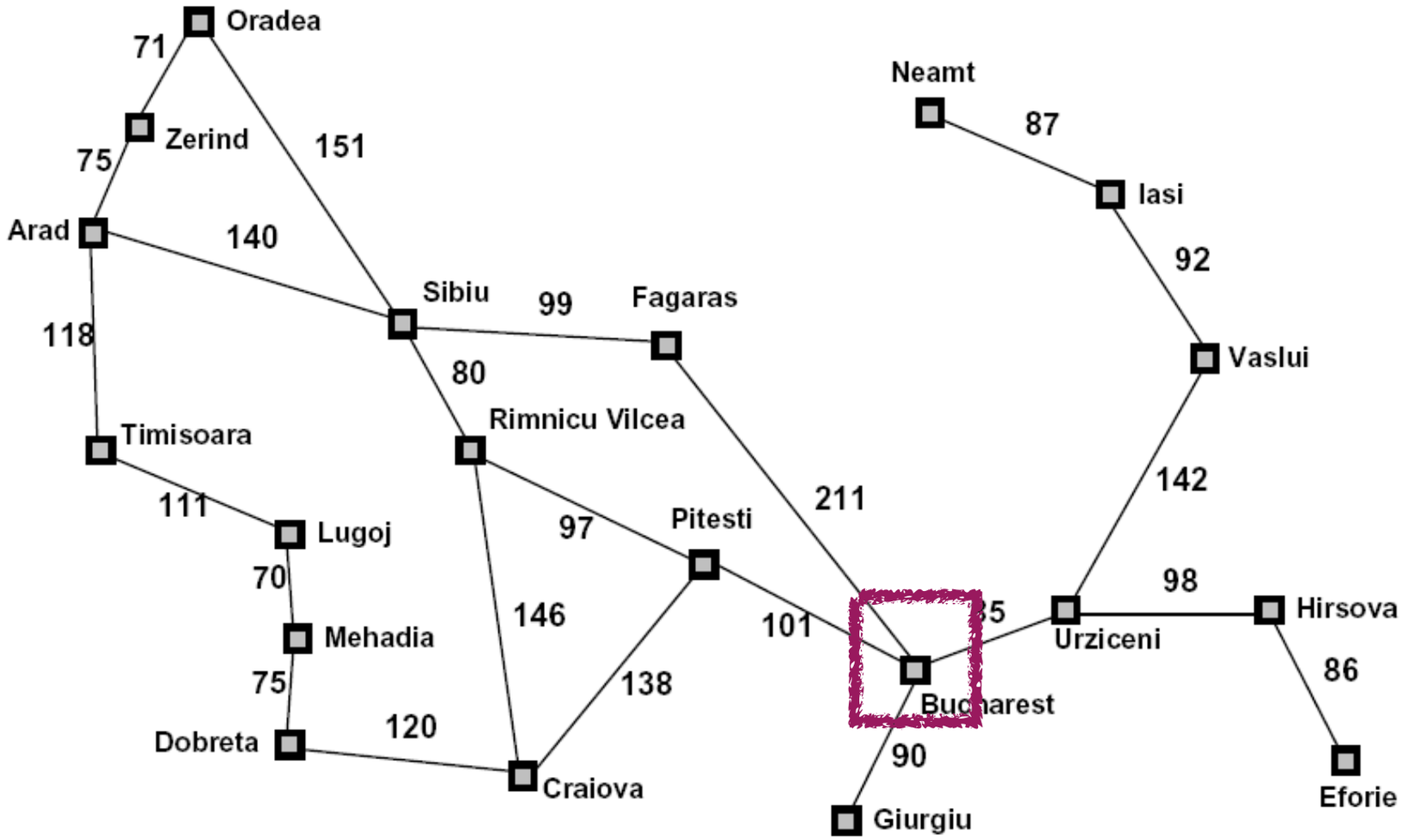
- Uninformed search
  - Breadth-first search
  - Depth-first search
- Informed search
  - Best-first search
  - A* search
- Take-Home Messages

The Limitation of Uninformed Search



The algorithm is short-sighted: no information about the long-term goal ahead.

Benefit of Look-Ahead Information?



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

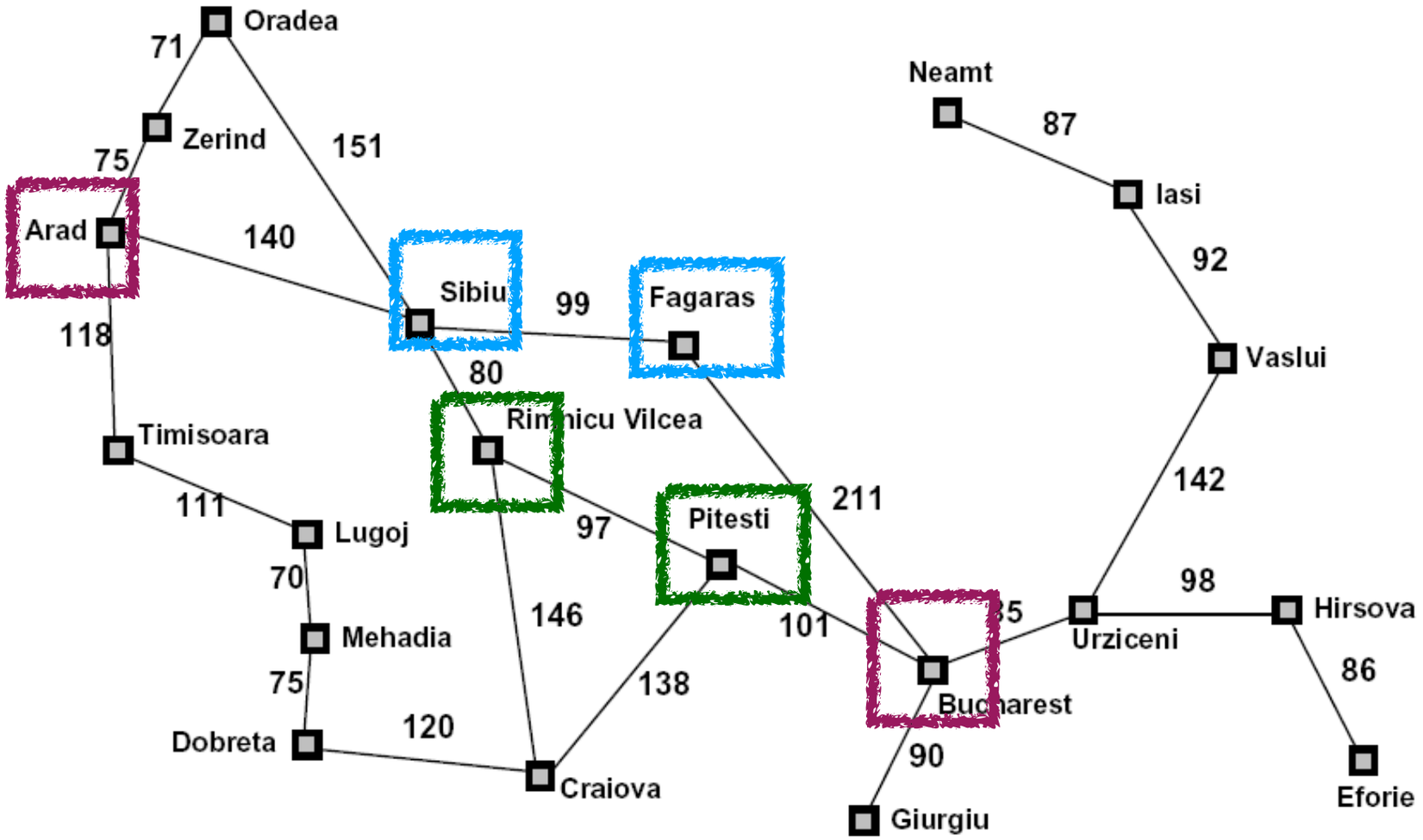
heuristic function

What if knowing something (heuristic function) about the goal during search?

Outline: Decision Making (I)

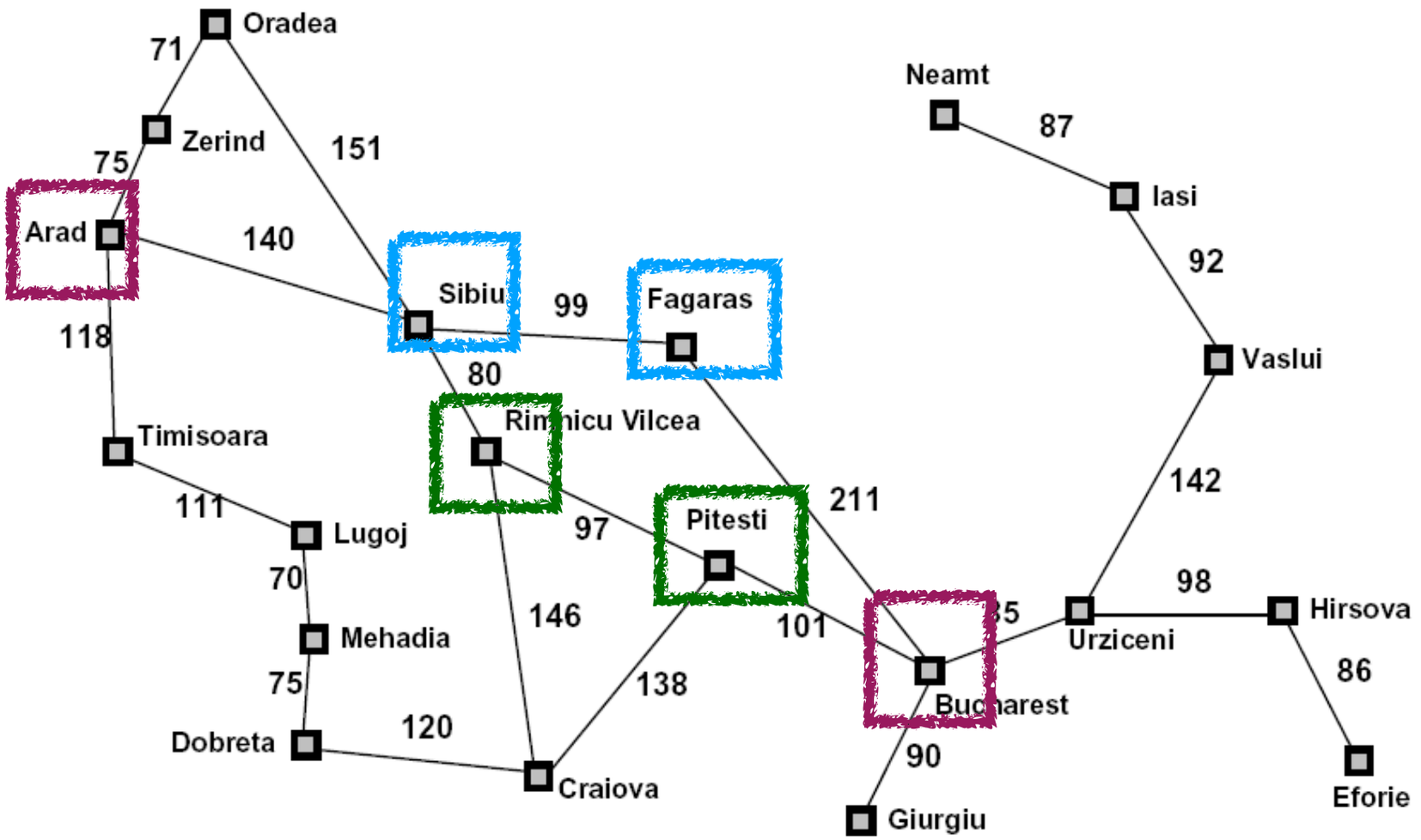
- Uninformed search
  - Breadth-first search
  - Depth-first search
- Informed search
  - Best-first search
  - A* search
- Take-Home Messages

Best-First Search (Greedy Search)



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

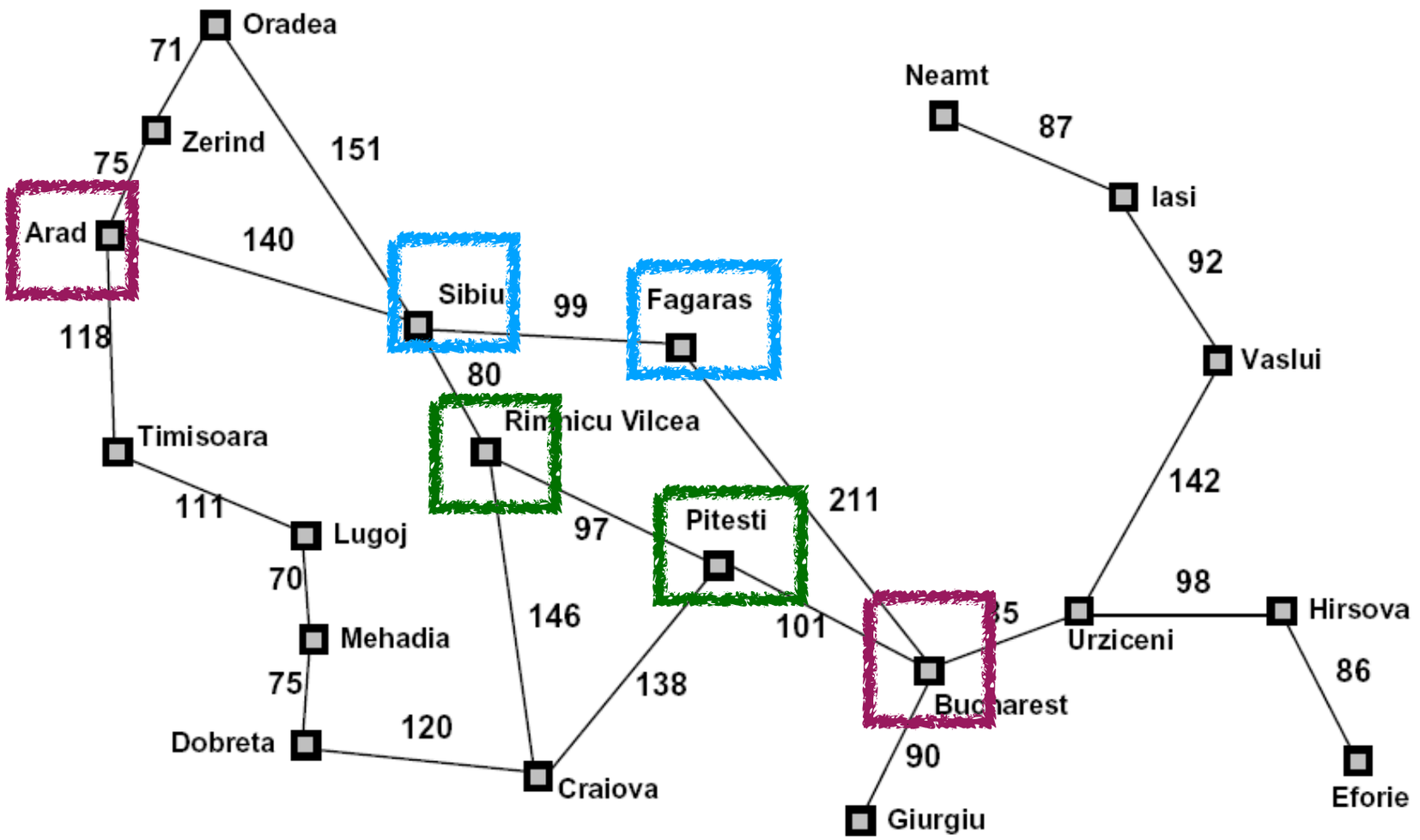
Best-First Search (Greedy Search)



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Always expand the node with the best heuristic function value

Best-First Search (Greedy Search)



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Always expand the node with the best heuristic function value

As usual, greedy strategy can lead to sub-optimal results.

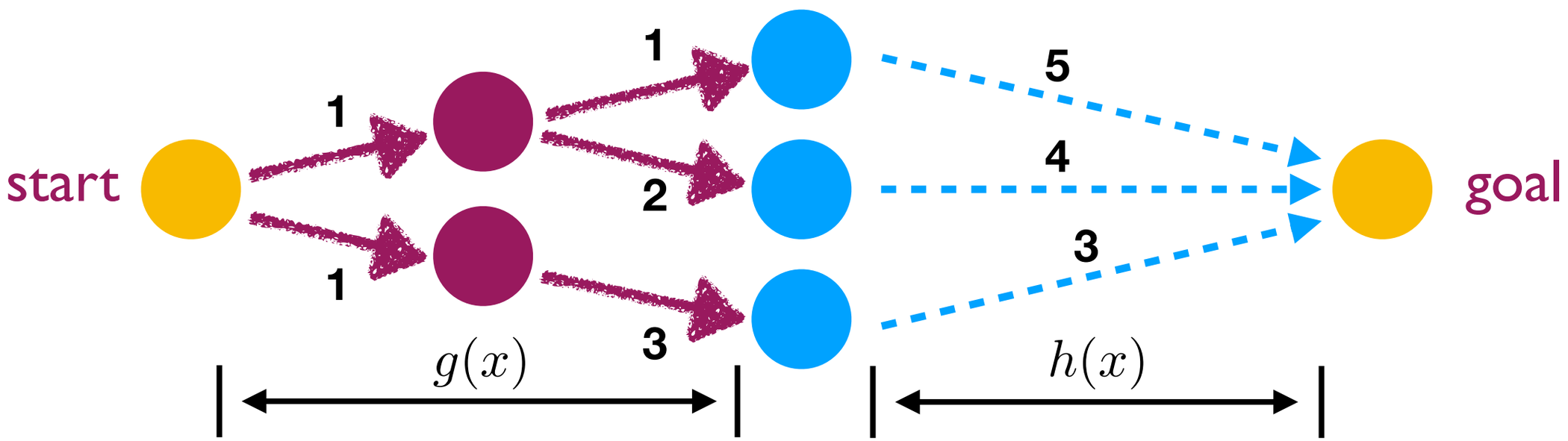
Best-First Search (Greedy Search)

Actually, greedy is sub-optimal since we cannot have a perfect heuristic which is hard to obtain in practice.
We need an algorithm that works under good heuristics other than perfect.

Outline: Decision Making (I)

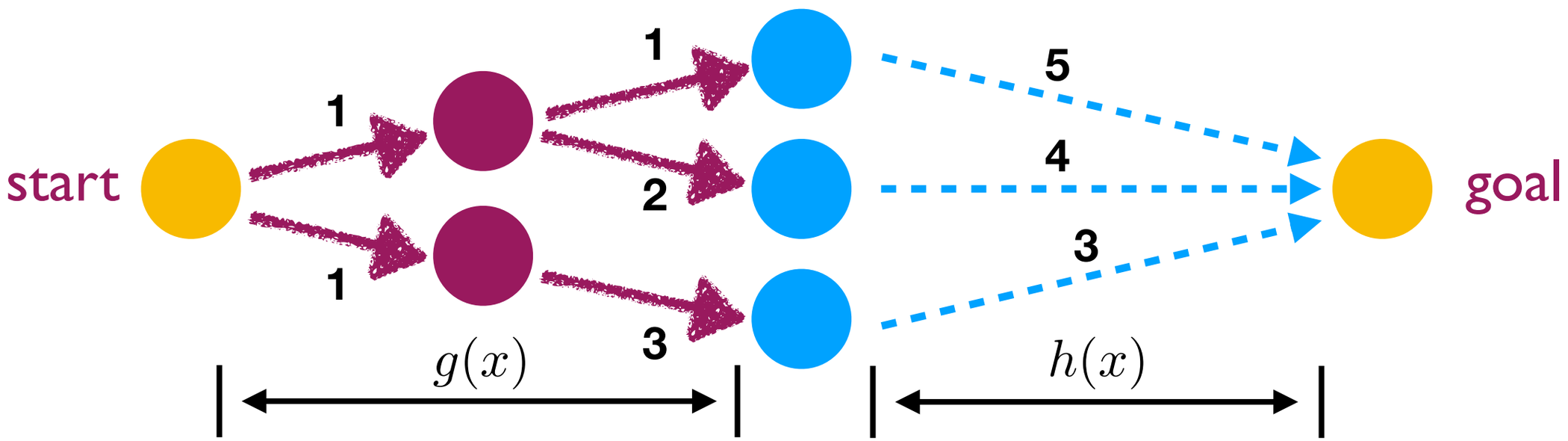
- Uninformed search
  - Breadth-first search
  - Depth-first search
- Informed search
  - Best-first search
  - A* search
- Take-Home Messages

A* Search



- A* chooses one x to expand with minimum $f(x) = g(x) + h(x)$
- $g(x)$: known costs from the start to the nodes.
- $h(x)$: heuristic to estimate the costs from the nodes to the goal.

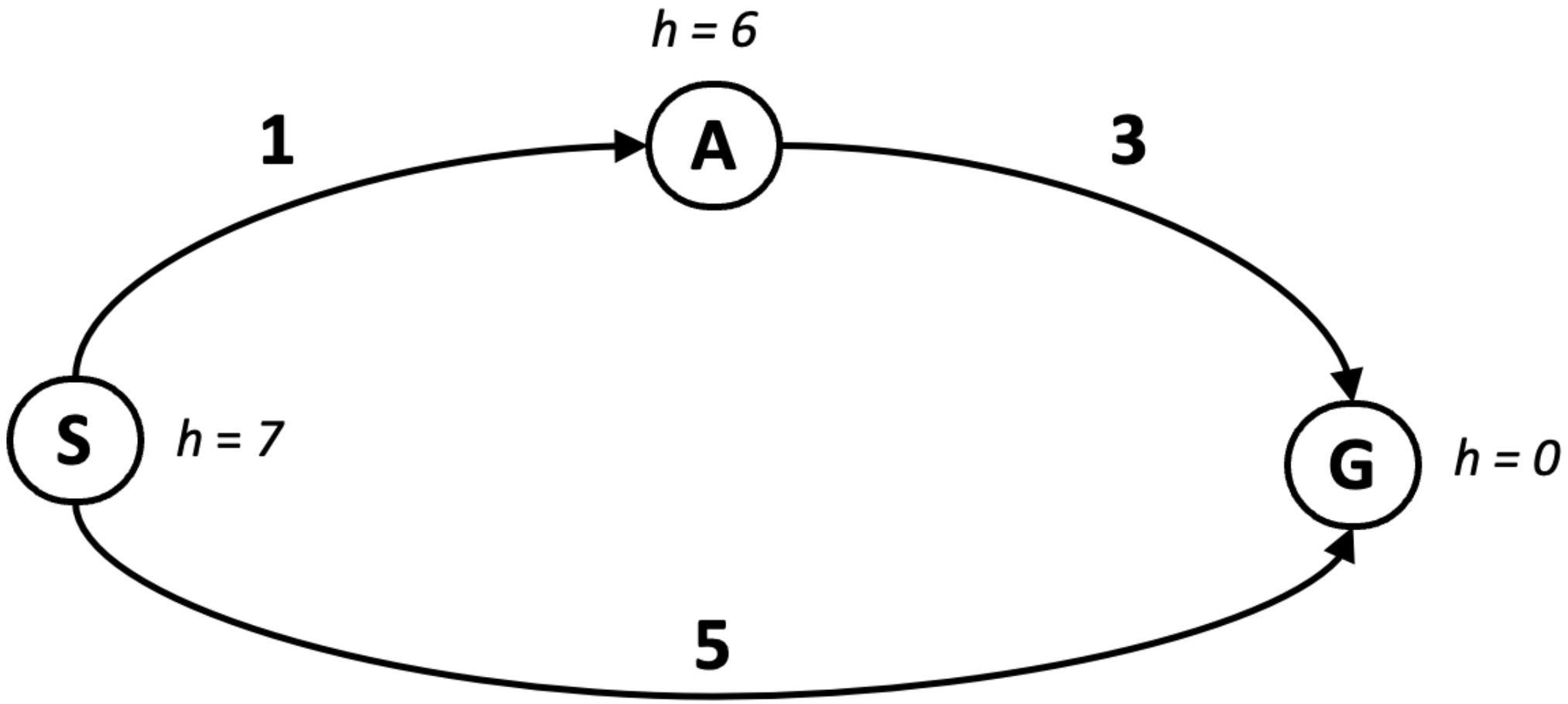
A* Search



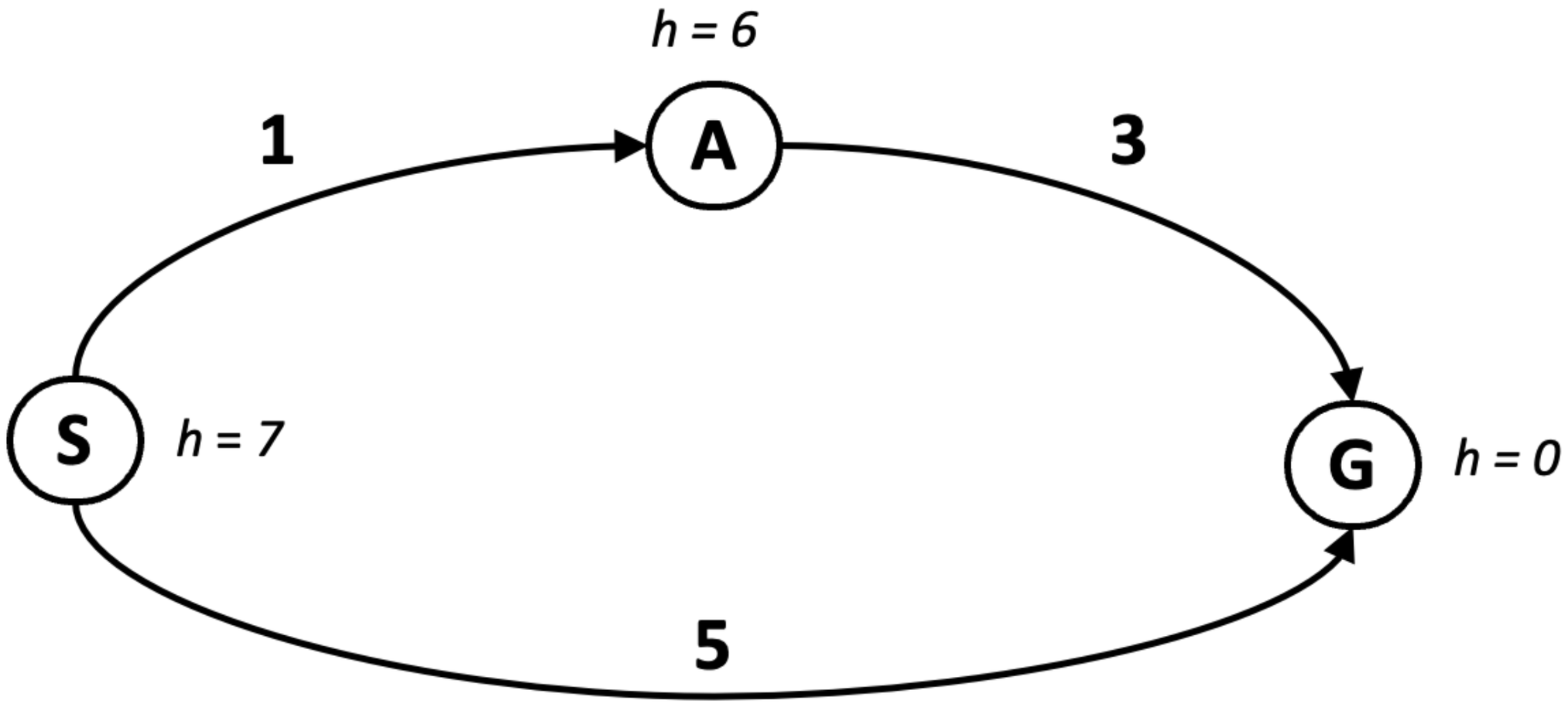
- A* chooses one x to expand with minimum $f(x) = g(x) + h(x)$
- $g(x)$: known costs from the start to the nodes.
- $h(x)$: heuristic to estimate the costs from the nodes to the goal.

Combination of UCS and greedy.

Will A* Find the Optimal Solution?



Will A* Find the Optimal Solution?

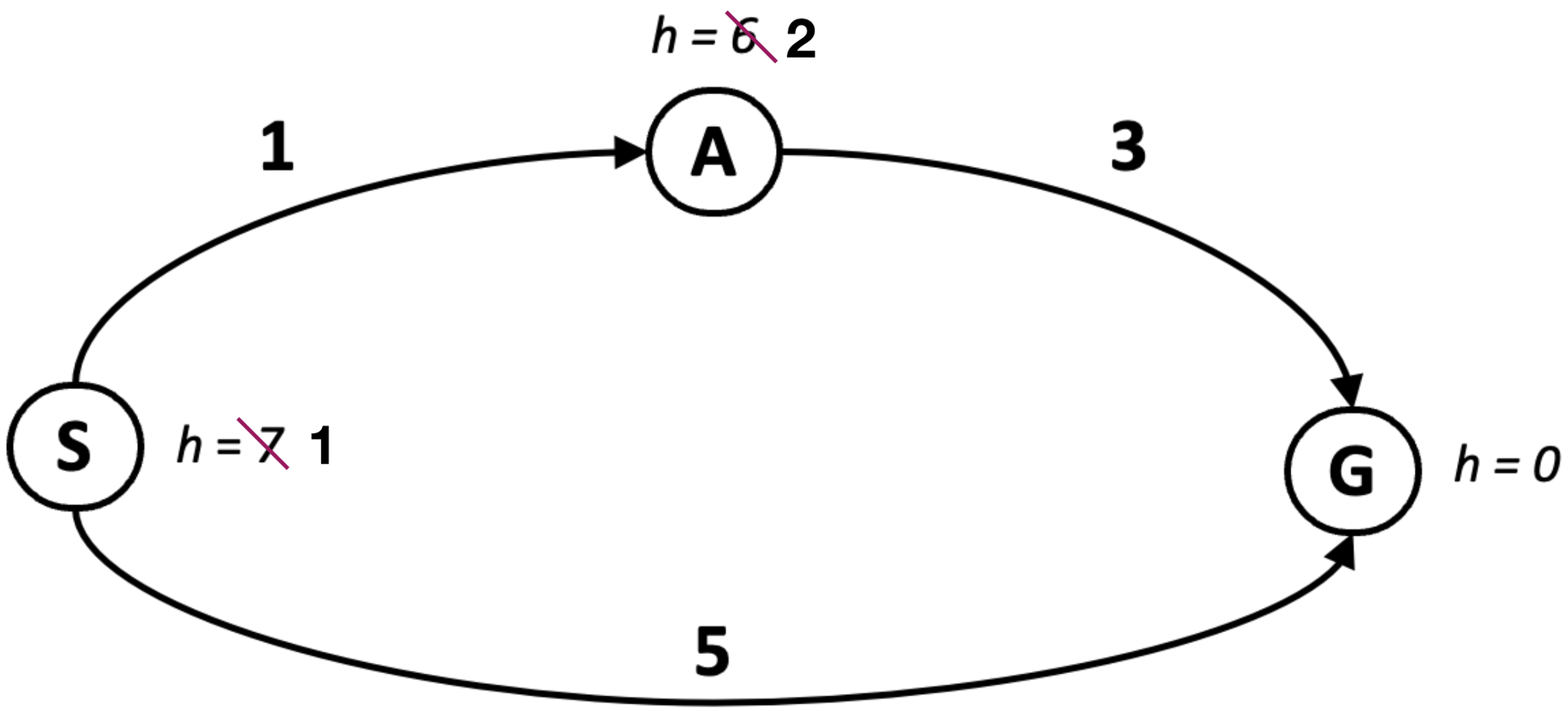


Additional assumptions are necessary.
The key is not to ignore good nodes.
We shouldn't over-estimate good node costs!

Admissible Heuristics

- A heuristic function h is called admissible if for any node n,

$$0 \leq h(n) \leq cost(n, G)$$

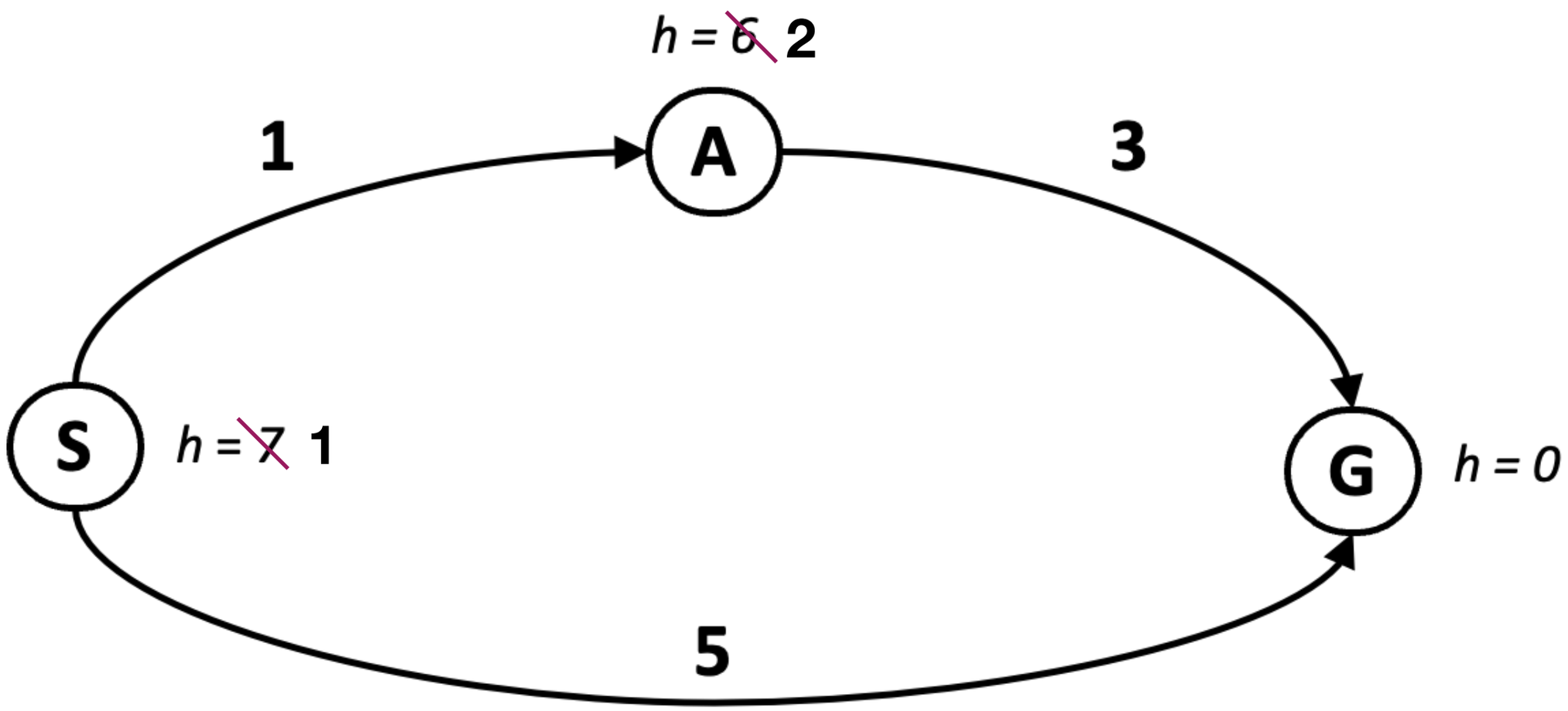


Admissible Heuristics

- A heuristic function h is called admissible if for any node n,

$$0 \leq h(n) \leq cost(n, G)$$

Admissible heuristics are always optimistic (even for bad nodes).



Optimality of A*

Theorem:
A* guarantees to find (one of) the optimal goal when admissible heuristic function is used.

Optimality of A*

Theorem:
A* guarantees to find (one of) the optimal goal when admissible heuristic function is used.

Proof:
Prove that for any bad goal state, the nodes on the path to the optimal goal will be expanded before it.

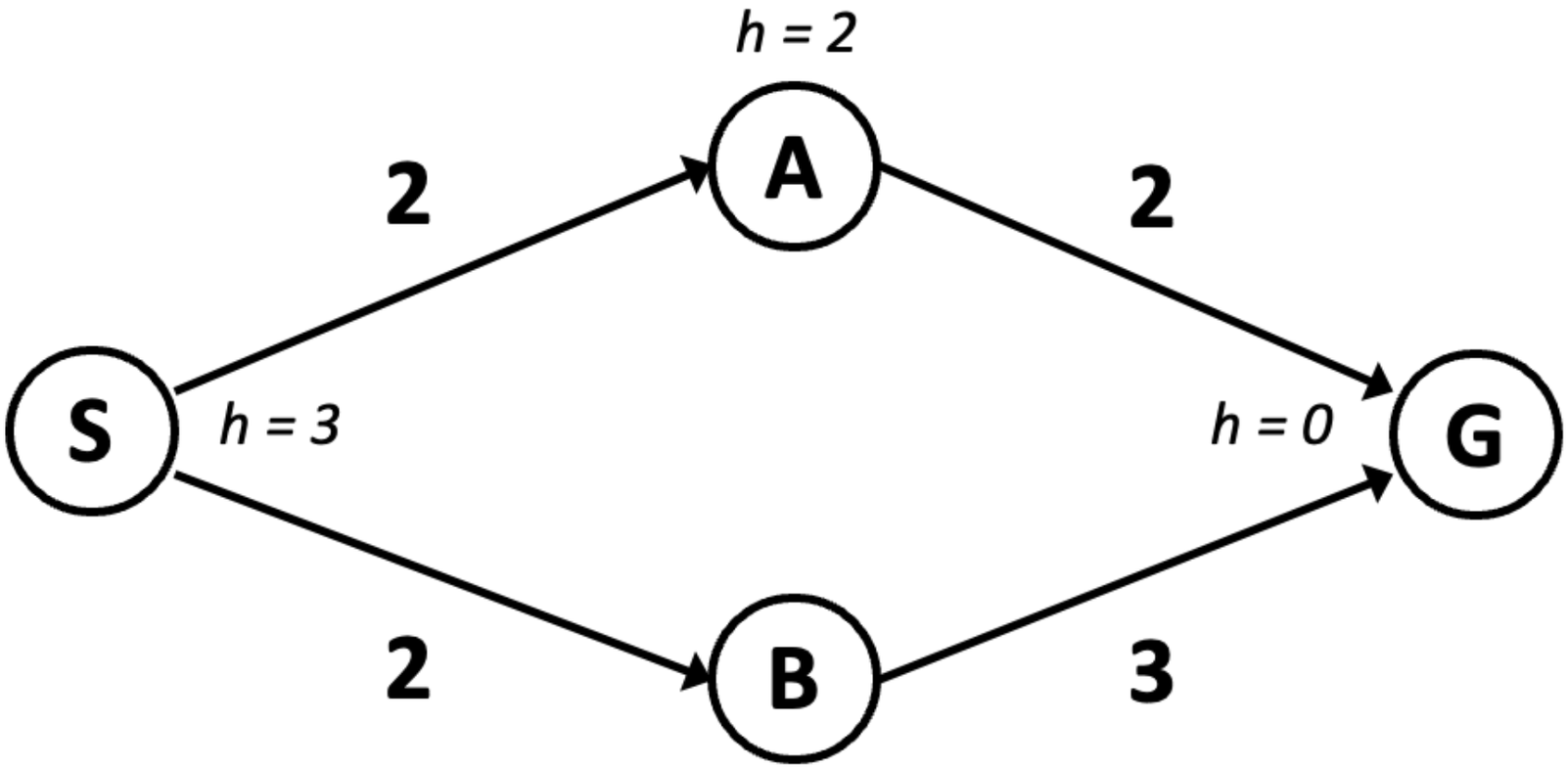
Optimality of A*

Theorem:
A* guarantees to find (one of) the optimal goal when admissible heuristic function is used.

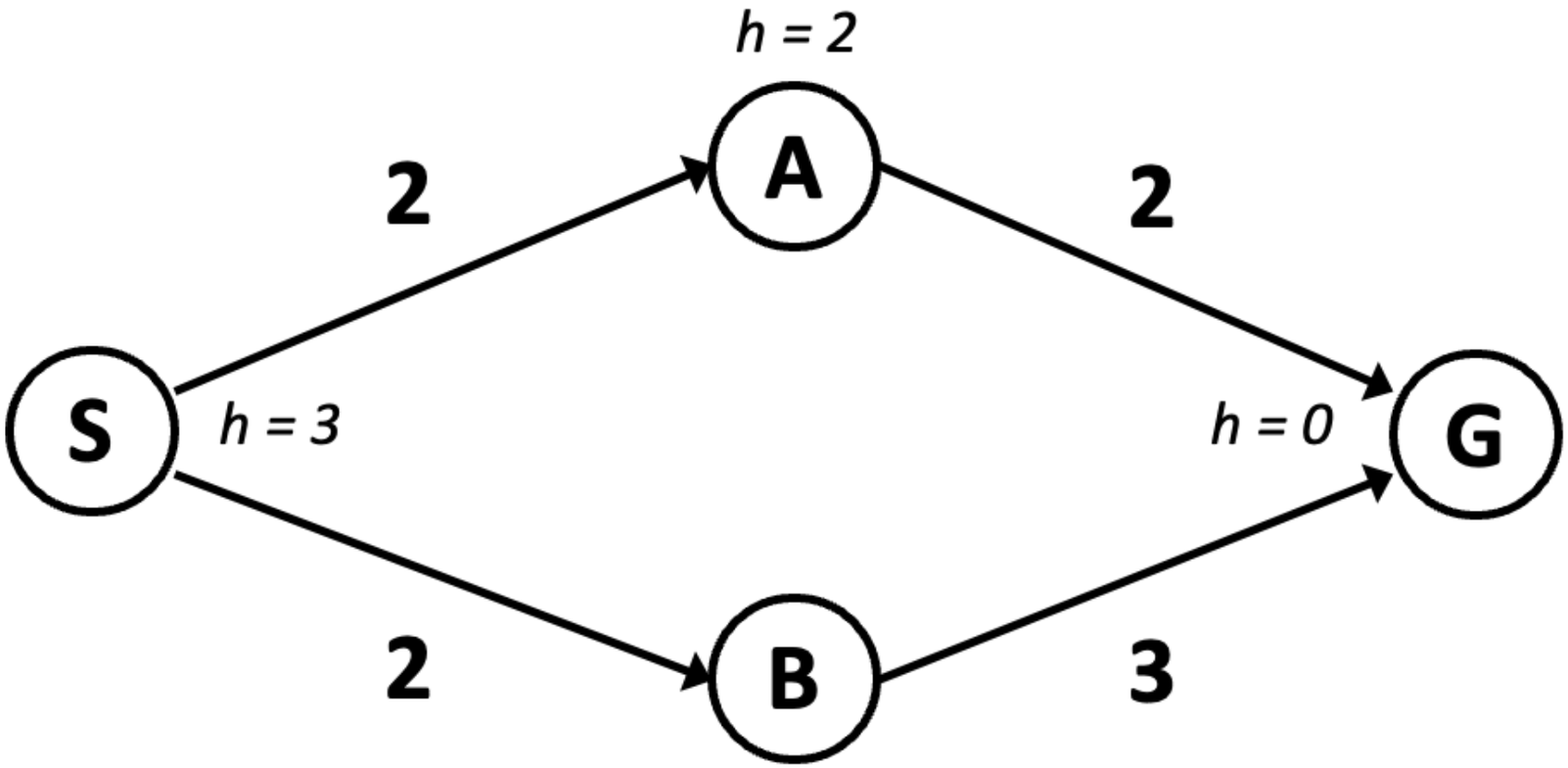
Proof:
Prove that for any bad goal state, the nodes on the path to the optimal goal will be expanded before it.

Can A* find the optimal path to the optimal goal?

Can A* Find the Optimal Path?



Can A* Find the Optimal Path?



To find the optimal path,
we should stop the search when the goal is dequeued
other than when it is firstly found.

A* Search Applications

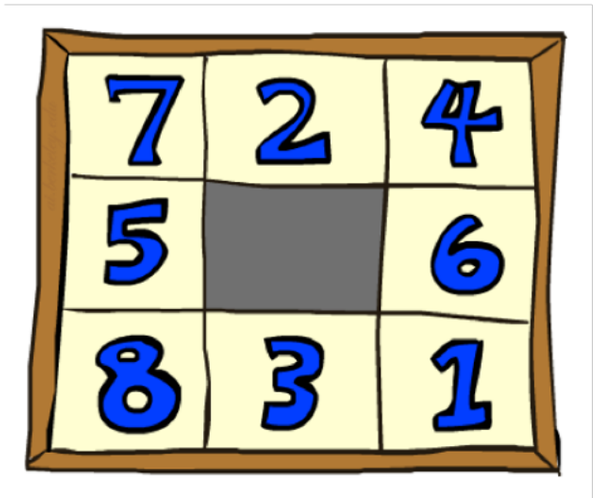
- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

A* Search Applications

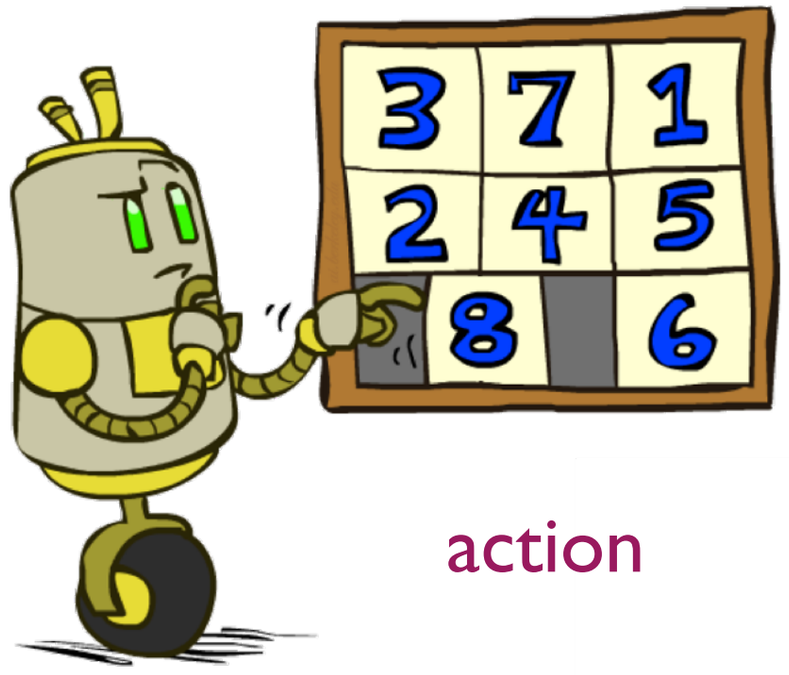
- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

Why so useful?
Design of heuristic functions :-P

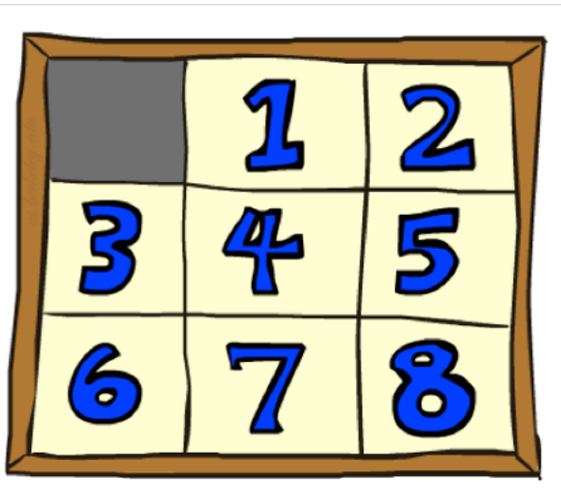
Design Admissible Heuristics



start



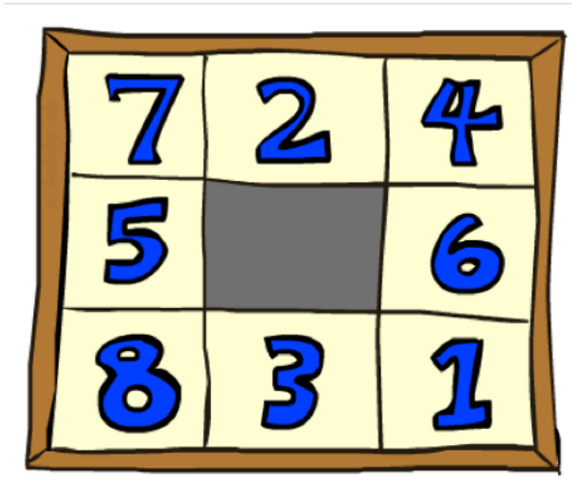
action



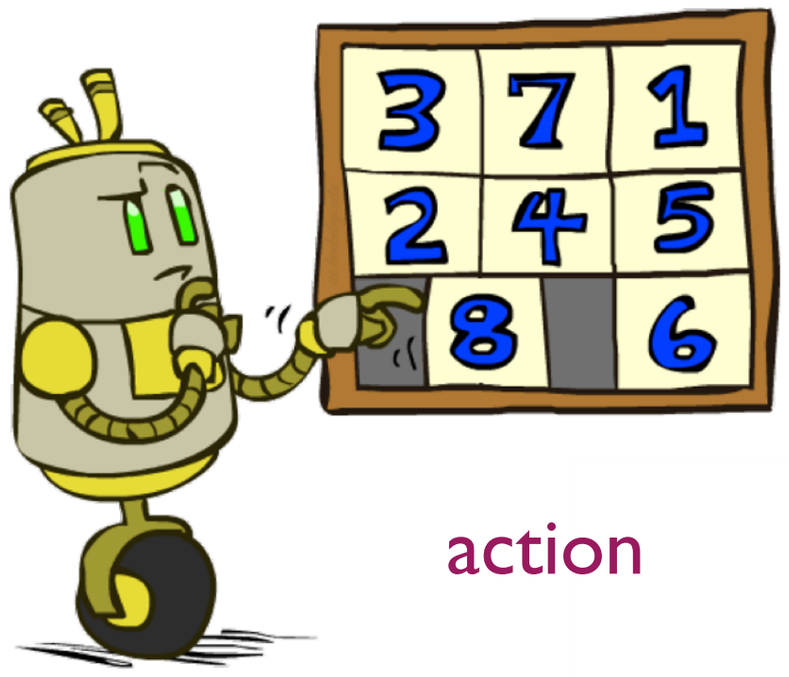
goal

- The game of 8 puzzle:
  - What is the state space?
  - How many states?
  - What is the action and cost?

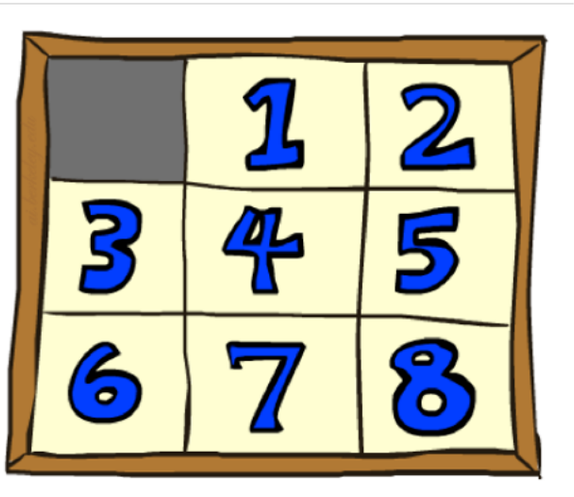
Design Admissible Heuristics



start



action



goal

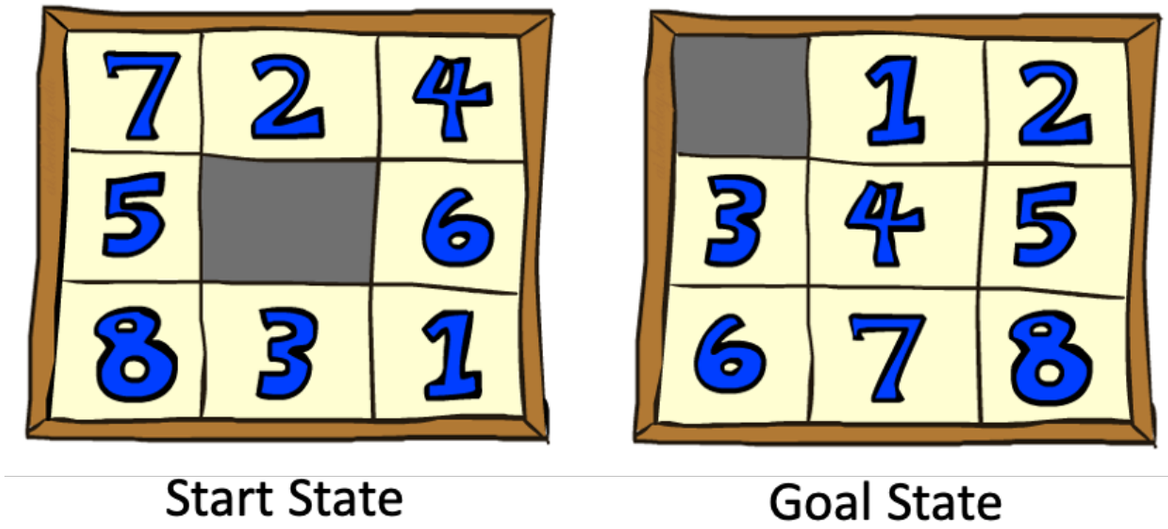
- The game of 8 puzzle:
  - What is the state space?
  - How many states?
  - What is the action and cost?

Any idea for
admissible heuristic?

Design Admissible Heuristics

- Admissible heuristics:
  - The number of misplaced pieces
  - Total Manhattan distance:

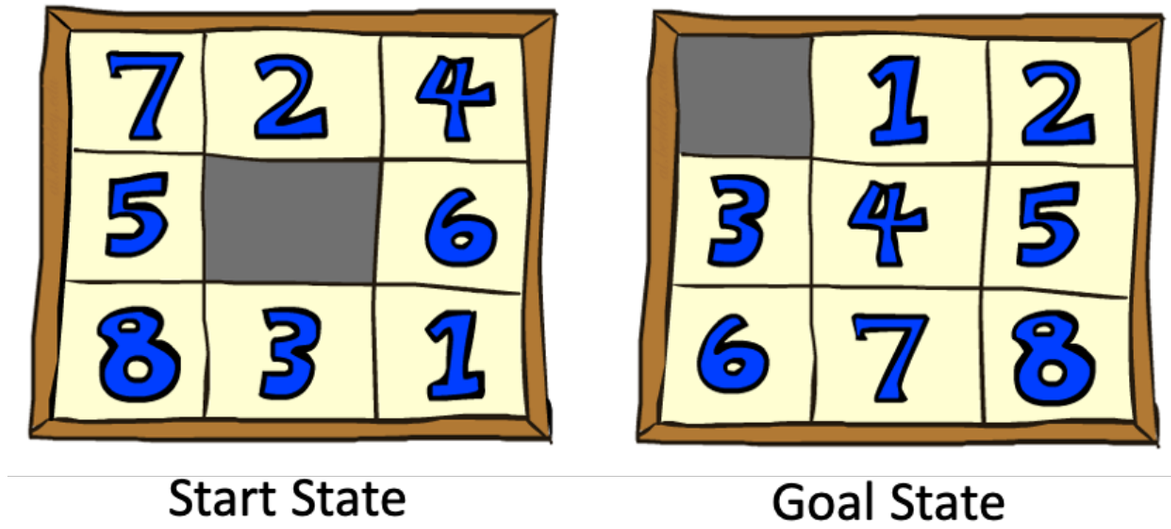
$$d((x_1, y_1), (x_2, y_2)) = |x_1 - y_1| + |x_2 - y_2|$$



Design Admissible Heuristics

- Admissible heuristics:
  - The number of misplaced pieces
  - Total Manhattan distance:

$$d((x_1, y_1), (x_2, y_2)) = |x_1 - y_1| + |x_2 - y_2|$$

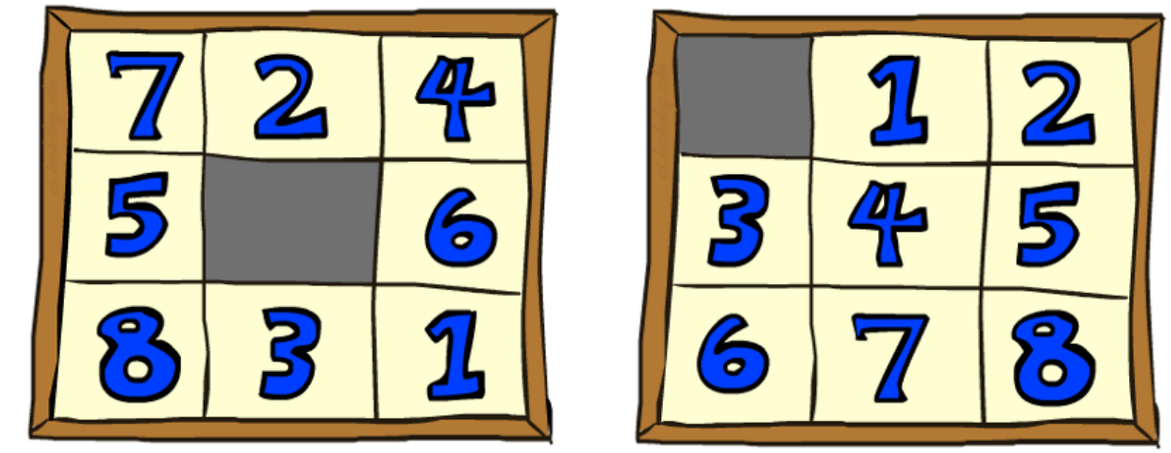


How about use $h(n) = 0$?
How about use the true cost as the heuristic?

Design Admissible Heuristics

- Admissible heuristics:
  - The number of misplaced pieces
  - Total Manhattan distance:

$$d((x_1, y_1), (x_2, y_2)) = |x_1 - y_1| + |x_2 - y_2|$$

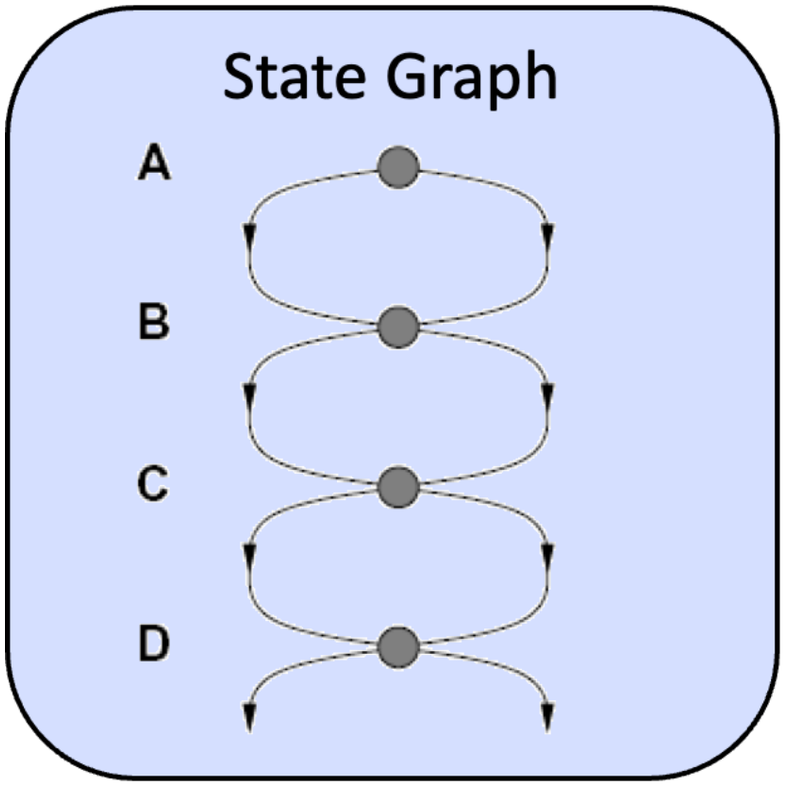


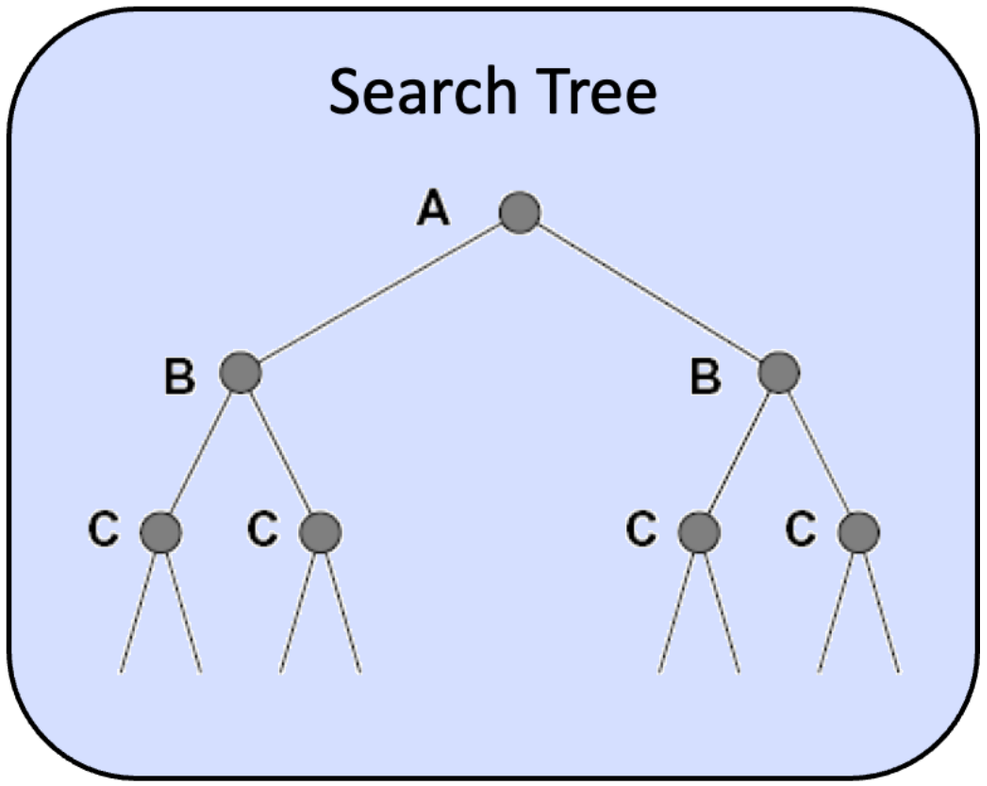
Start State Goal State

How about use $h(n) = 0$?
How about use the true cost as the heuristic?

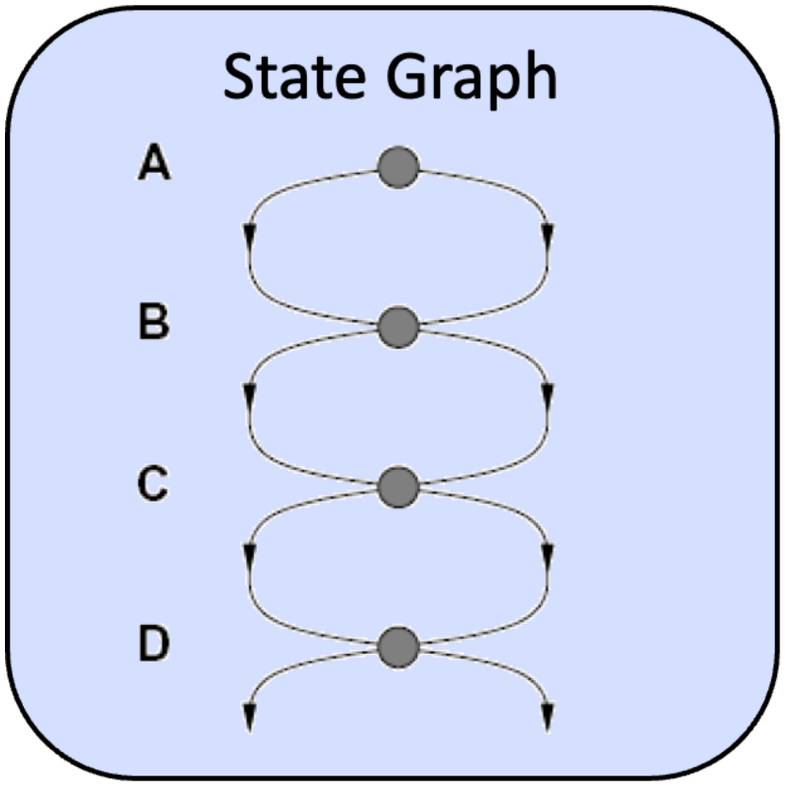
The first is equivalent to UCS.
The second is not possible since we should obtain the true cost.
Useful heuristics are between these two extremes!

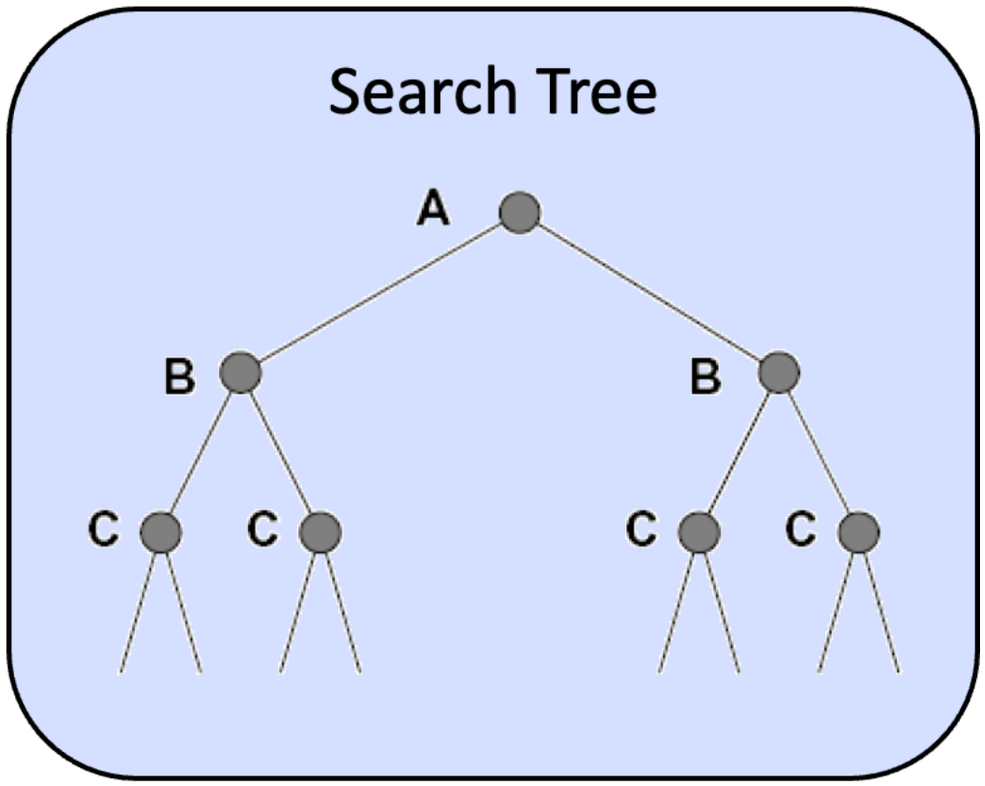
Tree Search vs. Graph Search





Tree Search vs. Graph Search

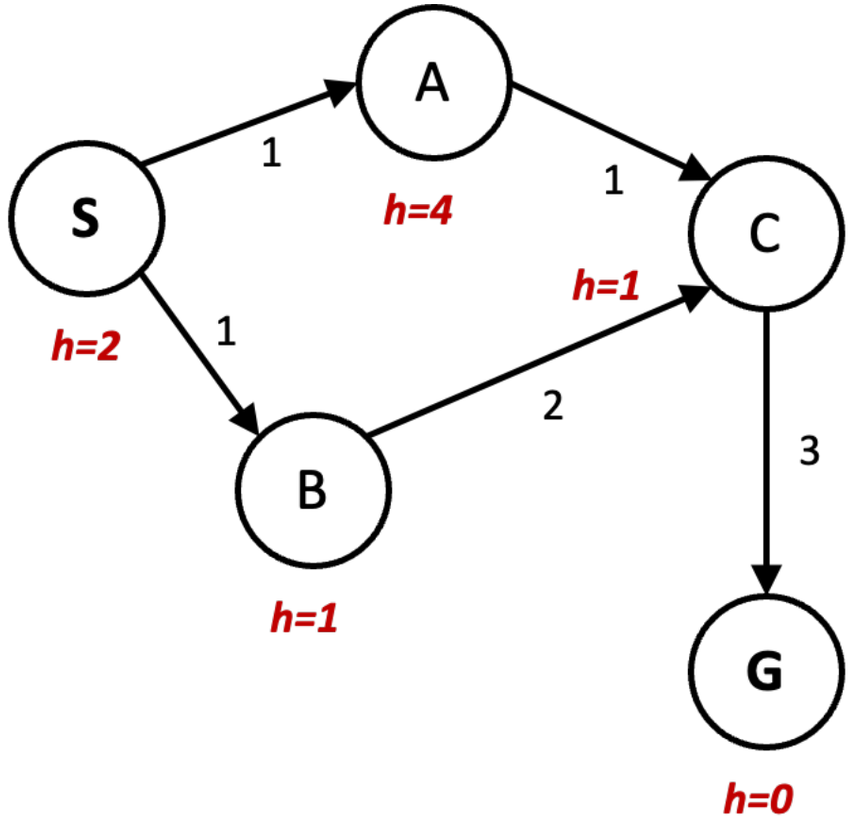




For many state graphs, there are many duplicated nodes in the search tree.
Direct tree search may be wasteful.

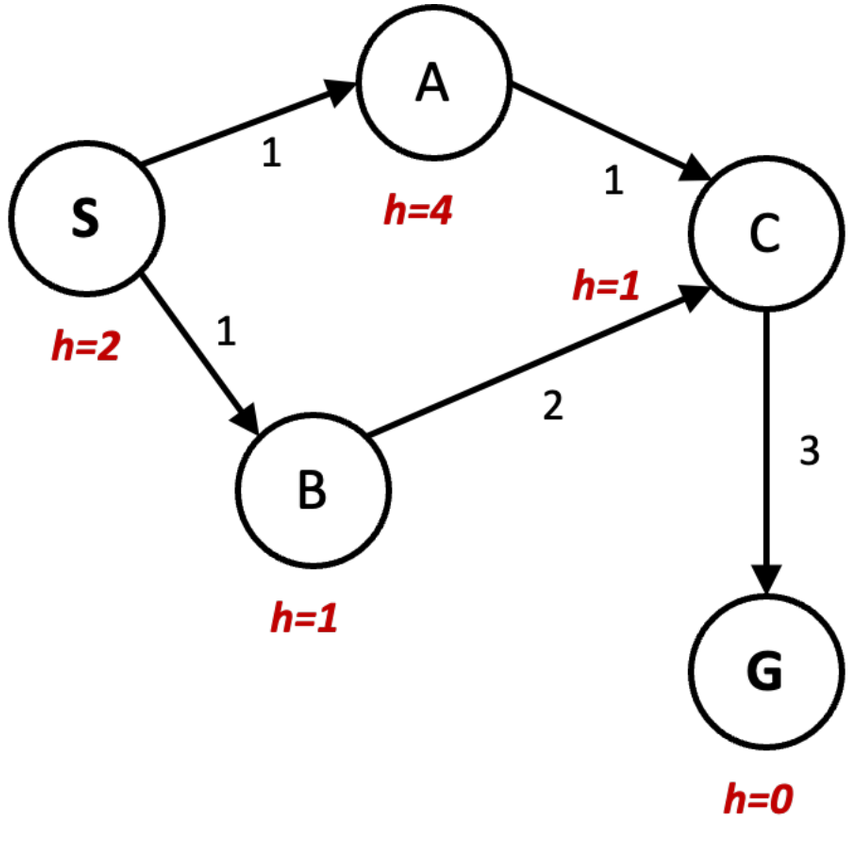
Graph Search

- Idea: never expand a node twice.
- Maintain a closed set of visited nodes.
- A node will not be expanded if it is already in the closed set.



Graph Search

- Idea: never expand a node twice.
- Maintain a closed set of visited nodes.
- A node will not be expanded if it is already in the closed set.

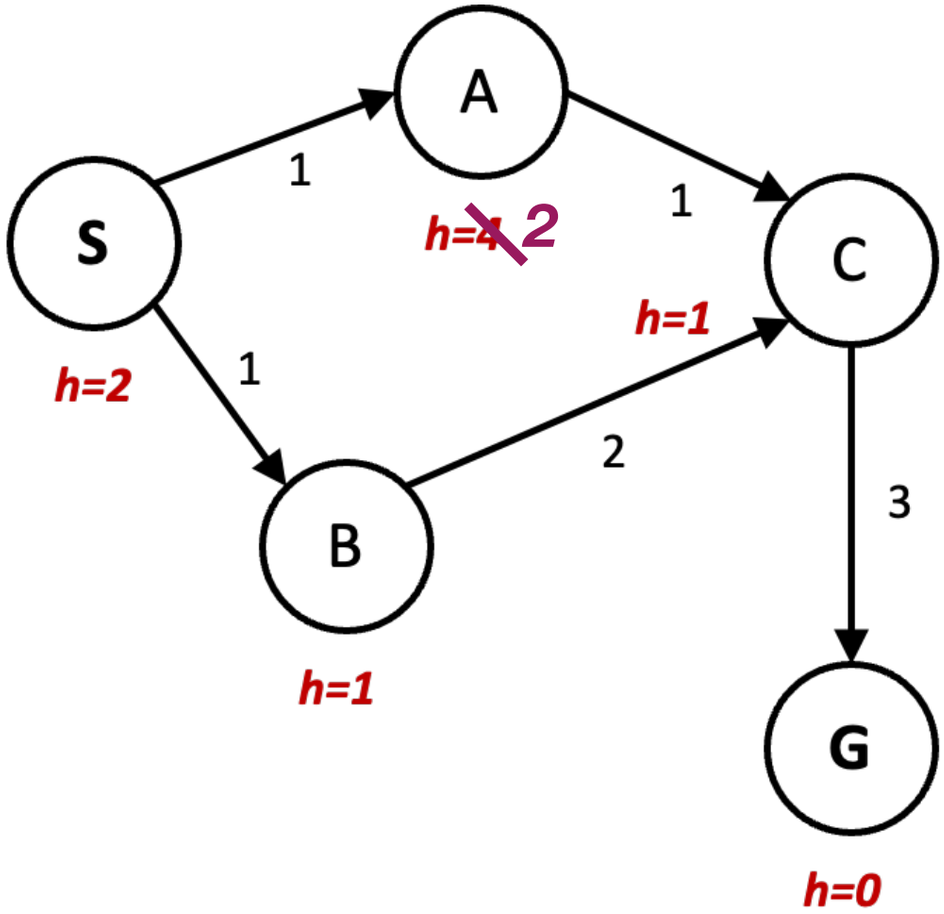


For graph search, A* may not be optimal under admissibility.

Consistency of Heuristic Functions

- We need a strong requirement of the heuristic function.
  - Guarantee to reach ~~the goal~~ any node in the right way.
  - Consistency: For any two nodes n, n',

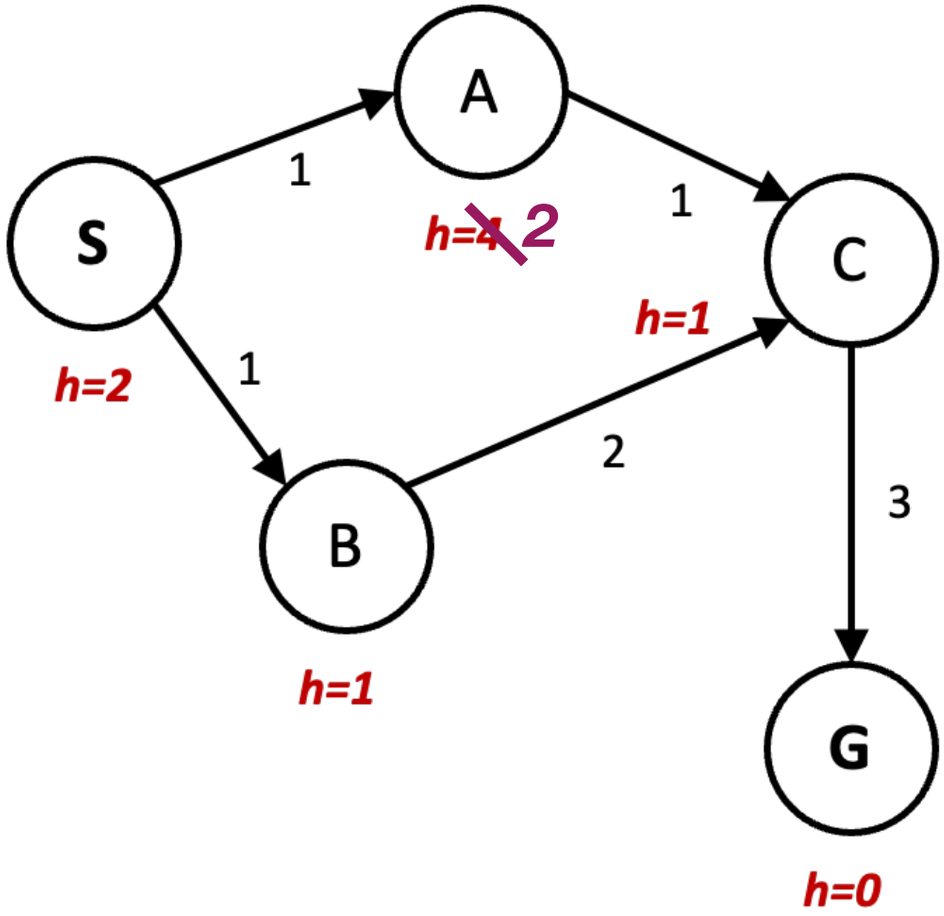
$$h(n) \leq cost(n, n') + h(n')$$



Consistency of Heuristic Functions

- We need a strong requirement of the heuristic function.
  - Guarantee to reach ~~the goal~~ any node in the right way.
  - Consistency: For any two nodes n, n',

$$h(n) \leq cost(n, n') + h(n')$$



Obviously, consistency implies admissibility.

Optimality of Graph-Search A*

Theorem:
Graph search A* guarantees to find (one of) the optimal goal when consistent heuristic function is used.

Optimality of Graph-Search A*

Theorem:
Graph search A* guarantees to find (one of) the optimal goal when consistent heuristic function is used.

Proof:
Can you derive it based on the admissible proof? :-P

Outline: Decision Making (I)

- Uninformed search
  - Breadth-first search
  - Depth-first search
- Informed search
  - Best-first search
  - A* search
- Take-Home Messages

Take-Home Messages

- Problem-solving agents
  - Aim at reaching a goal and minimizing the total cost
  - Use the external model given by the environment
- Uninformed search
  - BFS & DFS: without using additional information about the goal.
- Informed search (A*):
  - Looking backward and forward simultaneously.
  - Use admissible or consistent heuristic functions.

Thanks for your attention!
Discussions?

Acknowledgement: Many materials in this lecture are taken from
http://ai.berkeley.edu/lecture_slides.html